Knowledge representation with multiple logical theories and time

PAOLO MANCARELLA, ALESSANDRA RAFFAETÀ
and FRANCO TURINI

Dipartimento di Informatica, Università di Pisa,
Corso Italia, 40, I-56125 Pisa, Italy
email: {paolo,raffaeta,turini}@di.unipi.it

Abstract. We present a knowledge representation framework where a collection of logic programs can be combined together by means of meta-level program composition operations. Each object-level program is composed of a collection of extended clauses, equipped with a time interval representing the time period in which they hold. The interaction between program composition operations and time yields a powerful knowledge representation language in which many applications can be naturally developed. The language is given a meta-level semantics which also provides an executable specification. Moreover, we define an abstract semantics by extending the immediate consequence operator from a single logic program to compositions of logic programs and taking into account time intervals. The operational, meta-level semantics is proven sound and complete with respect to the abstract bottom-up semantics. The approach is further extended in order to cope with the problem of reasoning over joined intervals of time. Three applications in the field of business regulations are shown.

Keywords: logic programming, meta-logic, temporal reasoning, knowledge representation.

1. Introduction

Logic programming has been widely recognized as a powerful knowledge representation tool in various computing domains. It can be used both for procedural and declarative knowledge representation, and consequently it can be used for both programming and program specification, database applications and for knowledge representation and problem solving in artificial intelligence.

The development of logic programming, however, has shown that the basic paradigm is not expressive enough to deal naturally with several computing problems. To overcome some of these limitations, many extensions of logic programming have been studied to improve its knowledge representation and problem solving capabilities, such as the ability of handling negation, constraints, and abstraction mechanisms.

In this paper we propose yet another extension which addresses the handling of temporal information. Even though there are many proposals in the literature in the field of both temporal databases (see, for example, Tansel et al. (1993)) and

0952–813X/99 $12·00

temporal logic languages (see, e.g. Interval Temporal Logic (Allen 1984), Event Calculus (Kowalski and Sergot 1986), $Datalog_{1S}$ (Chomicki and Imielinski 1988), Temporal Prolog (Gabbay 1987, Hrycej 1993), and Templog (Abadi and Manna 1989)), our proposal addresses the handling of time dependent knowledge within the multi-theory framework presented in Brogi et al. (1994). This framework allows one to represent knowledge as separate logic theories, which can be combined together by means of various meta-level operators. From a deductive database perspective, each logic program (theory) can be viewed as an extended relational database where relations are represented partly intensionally and partly extensionally. The meta-level operators can then be viewed as a means of constructing views by combining multiple databases in various ways.

The need for extending the multi-theory framework with time came out clearly when we addressed knowledge representation problems, especially in the field of business data and procedures, and regulations and laws. A simple example may be useful to clarify the critical points. Consider the problem of representing the fact that a movie ticket is \$5 for kids and \$7 for adults in the first 6 months of 1996 and \$7 for everybody in the rest of the year. We allow one to attach time intervals to a clause, representing the period of time during which the clause is valid. The following theory *BoxOffice* adopts this representation to model the above information about movie tickets.

BoxOffice:

ticket-cost(5,p) ←
  age(p,a), a ⩽ 16 □
  [<Jan 1 1996>,<Jun 30 1996>]

ticket-cost(7,p) ←
  age(p,a), a > 16 □
  [<Jan 1 1996>,<Jun 30 1996>]

ticket-cost(7,p) ←
  age(p,-) □
  [<Jul 1 1996>,<Dec 31 1996>]

age(p,a) ←
  today(d1), born(p,d2), year-diff(d2,d1,a) □
  [<Jan 1 1900>,α]

where α stands for a time point later than any other one, and *year-diff*(*d2,d1,a*) computes the age *a* given the current date *d*1 and the date of birth *d*2. The last clause for *ticket-cost* represents the fact that the ticket is \$7 in the second part of the year, regardless of the customer's age. Moreover, notice that the theory is *parametric* with respect to the actual day of the year and the customer's birthday. The predicate *today* is defined in a separate theory which represents the current date, e.g.

Today:

today(<May 28 1996>) ←□
  [<May 28 1996>,<May 28 1996>]

and the customer's birthday is given in a separate theory like

Tom:

born(Tom,<May 7 1981>) ←□
  [<May 7 1981>,α]

The previous theories can be combined by means of a union operator ⋃ (see section 2): the query *ticket-cost(s, Tom)* with respect to the combined knowledge

$$BoxOffice \bigcup Tom \bigcup Today$$

yields the answer $s = 5$.

As shown in the example, the kinds of applications we are interested in suggest the following decisions about the introduction of time in our framework:

- the primitive notion of time is *an interval of time*,
- time intervals are attached to clauses.

Of course, there are many other options, e.g. using time points and relations among them, attaching time information to relations instead of rules and so on. Investigating the impact of these alternative choices on our framework is one of the tasks we intend to pursue in the immediate future.

The paper is organized as follows. Section 2 briefly introduces the operators for combining logic theories. Meta-logic is exploited to provide a formal, and, at the same time, executable semantics to the operators. Section 3 discusses the introduction of time intervals and its semantics, still based on meta-logic. Section 4 provides our language with an abstract semantics by extending the definition of the immediate consequence operator from a single logic program to expressions of logic programs. This section includes the proofs of the soundness and completeness of the operational semantics, given by means of meta-logics in section 3, with respect to the abstract semantics. Section 5 deals with the application of the framework discussed so far to representing knowledge in the field of regulations and section 6 introduces some auxiliary operators to make easier the handling of time. Section 7 addresses the problem of handling reasoning over joined intervals, that provides the possibility of computing the maximal interval in which an answer holds. Section 8 presents various proposals in the literature dealing with time and finally, section 9 outlines our future research plans. An extended abstract of this paper was presented at TIME'97 (Mancarella et al. 1997).

2. Operators for combining logic theories

Program composition operations have been thoroughly investigated in Brogi (1993) and Brogi et al. (1994), where both their meta-level and their bottom-up semantics are studied and compared. Here, we adopt the meta-level definition of the operations, which is simply obtained by adding new clauses to the well-known vanilla meta-interpreter for logic programs. Stated otherwise, in this view compositions of programs are realized by a meta-interpreter which combines separate programs at the meta-level, without actually building a new program. The reading of the resulting meta-interpreter is straightforward and, most importantly, the meta-logical definition shows that the multi-theory framework can be expressed from inside logic programming itself. We provide two operators to combine programs: union $\bigcup$ and intersection $\bigcap$. Formally, we define the set of *program expressions Exp* with the following abstract syntax:

$$Exp ::= P \mid Exp \bigcup Exp \mid Exp \bigcap Exp$$

where P is a plain program, i.e. a collection of clauses.

Following Bowen and Kowalski (1982), we employ the two-argument predicate *demo* to represent provability. Namely, $demo(x, y)$ represents that the formula y is provable in the program expression x.

The *vanilla* meta-interpreter (Sterling and Shapiro 1986) is the simplest application

of meta-programming in logic. A general formulation of the vanilla meta-interpreter can be given by means of the *demo* predicate.

$$demo(x, empty) \leftarrow \tag{1}$$

$$demo(x, (y, z)) \leftarrow demo(x, y),\ demo(x, z) \tag{2}$$

$$demo(x, y) \leftarrow clause(x, y \leftarrow z),\ demo(x, z) \tag{3}$$

The unit clause (1) states that the empty goal, represented by the constant symbol *empty*, is solved in any program x. Clause (2) deals with conjunctive goals. It states that a conjunction (y, z) is solved in the program x if y is solved in x and z is solved in x. Finally, clause (3) deals with the case of atomic goal reduction. To solve an atomic goal y, a clause from the program x is chosen and the body of the clause is recursively solved in x.

We adopt the simple naming convention used in Kowalski and Kim (1991). Object programs are named by constant symbols, denoted by capital calligraphic letters such as $\mathcal{P}$ and $\mathcal{Q}$. Object level expressions are represented by themselves at the meta-level. In particular, object level variables are denoted by meta-level variables, according to the so-called *non-ground representation* (Hill and Lloyd 1989). An object level program $\mathcal{P}$ is represented at the meta-level by a set of axioms of the kind $clause(\mathcal{P}, A \leftarrow B) \leftarrow$, one for each object level clause $A \leftarrow B$ in $\mathcal{P}$. For example, we have the following object and meta-level representations, $\mathcal{N}$ and $\mathcal{N}_m$ respectively, of the logic program for natural numbers.

$\mathcal{N}$:
$nat(zero) \leftarrow$
$nat(s(x)) \leftarrow nat(x)$

$\mathcal{N}_m$:
$clause(\mathcal{N}, nat(zero) \leftarrow empty) \leftarrow$
$clause(\mathcal{N}, nat(s(x)) \leftarrow nat(x)) \leftarrow$

Program composition operations can be implemented by meta-logic in a simple and concise way. Each program composition operation is represented at the meta-level by a functor. The meaning of each functor is defined by new clauses added to the vanilla meta-interpreter.

$$clause(x \bigcup y, z \leftarrow w) \leftarrow clause(x, z \leftarrow w) \tag{4}$$

$$clause(x \bigcup y, z \leftarrow w) \leftarrow clause(y, z \leftarrow w) \tag{5}$$

$$clause(x \bigcap y, (z \leftarrow u, v)) \leftarrow clause(x, z \leftarrow u),\ clause(y, z \leftarrow v) \tag{6}$$

In the extended framework, the first argument of *clause* represents a program expression, rather than a single program as in the case of the pure vanilla meta-interpreter.

The meaning of the clauses (4)–(6) is straightforward. Informally, union and intersection mirror two forms of cooperation among program expressions. Clauses (4) and (5) define the meta-level implementation of the operation $\bigcup$, where either expression may be used to perform a computation step. For instance, a clause $z \leftarrow w$ belongs to the meta-level representation of $\mathcal{P} \bigcup \mathcal{Q}$ if it belongs either to the meta-level representation of $\mathcal{P}$ or to the meta-level representation of $\mathcal{Q}$. In the case of intersection, both expressions must agree to perform a computation step. This is obtained in clause (6) exploiting the basic unification mechanism of logic programming and the non-ground representation of object level programs. A program expression $\mathcal{E}$ can be queried by $demo(\mathcal{E}, G)$, where G is an object level goal.

3. Introducing time intervals

In this section we extend the multi-theory framework in order to handle temporal information. We associate time intervals to clauses of object-level programs. The meta-level representation of object-level programs must be extended accordingly as well as the meta-interpreter of the previous section.

Our temporal model can be classified as *historical* in the taxonomy of Snodgrass (1992). Time intervals represent *valid time*, that is the time for which information models reality and corresponds to the actual time for which a relationship holds in the real world. This allows us to query the database at a certain date in the past to obtain the information that held in that period.

We work on discrete time points. Actual time points are represented by elements of the set $\mathbb{N}$ of natural numbers. A special constant α is used to represent a time point later than any others. An interval is a pair $[a, b]$, where $a \in \mathbb{N}$ and $b \in \mathbb{N} \cup \{\alpha\}$. Notice that an interval open to the future is represented by $[a, \alpha]$. The order relation between time points is axiomatized as follows:

$$timepoint(0) \leftarrow$$
$$timepoint(s(x)) \leftarrow timepoint(x)$$
$$x < \alpha \leftarrow timepoint(x)$$
$$0 < s(x) \leftarrow timepoint(x)$$
$$s(x) < s(y) \leftarrow timepoint(x), timepoint(y), x < y$$

$$x \leqslant x \leftarrow$$
$$x \leqslant y \leftarrow x < y$$
$$x > y \leftarrow y < x$$
$$x \geqslant y \leftarrow y \leqslant x$$

The relation of inclusion between time intervals is denoted by $\sqsubseteq$ and defined as

$$[a, b] \sqsubseteq [c, d] \leftarrow non\text{-}empty([a, b]), non\text{-}empty([c, d]), a \geqslant c, b \leqslant d,$$

where the predicate *non-empty* states that an interval is not empty and it is defined as

$$non\text{-}empty([a, b]) \leftarrow a \leqslant b.$$

Moreover, intersection of time intervals is denoted by $\sqcap$ and it is axiomatized as follows:

$$[a, b] \sqcap [c, d] = [c, b] \leftarrow non\text{-}empty([a, b]), non\text{-}empty([c, d]), a \leqslant c, c \leqslant b, b \leqslant d$$
$$[a, b] \sqcap [c, d] = [c, d] \leftarrow non\text{-}empty([a, b]), non\text{-}empty([c, d]), a \leqslant c, d < b$$
$$[a, b] \sqcap [c, d] = [a, d] \leftarrow non\text{-}empty([a, b]), non\text{-}empty([c, d]), c \leqslant a, a \leqslant d, d \leqslant b$$
$$[a, b] \sqcap [c, d] = [a, b] \leftarrow non\text{-}empty([a, b]), non\text{-}empty([c, d]), c \leqslant a, b < d$$

Notice that intersection is defined only on *overlapping intervals*. Therefore when we consider an interval obtained by intersecting different intervals, i.e. $\prod_{i=1,n} I_i$, we mean it is not empty.

An object level program is still a collection of clauses named by a constant symbol. Each clause is now equipped with a time interval representing the period of time in which the clause holds. At the object level, an extended clause looks like

$$A \leftarrow B_1, \ldots, B_n \ \square \ [a, b].$$

According to the above extension, the meta-level interpreter of the previous section must be extended by taking time intervals into account. The predicate *demo* now has an extra-argument denoting a time interval: $demo(\mathscr{E}, G, I)$ means that the goal G holds with respect to the program expression $\mathscr{E}$ and within the time period I. The extended meta-interpreter is defined by the following clauses.

$$demo(x, empty, I) \leftarrow non\text{-}empty(I) \qquad (7)$$

$$demo(x,(y,z),I) \leftarrow demo(x,y,K), demo(x,z,J), I \sqsubseteq K \sqcap J \quad (8)$$

$$demo(x,y,I) \leftarrow clause(x, y \leftarrow z, K), demo(x,z,J), I \sqsubseteq K \sqcap J \quad (9)$$

$$clause(x \bigcup y, z \leftarrow w, I) \leftarrow clause(x, z \leftarrow w, K), I \sqsubseteq K \quad (10)$$

$$clause(x \bigcup y, z \leftarrow w, I) \leftarrow clause(y, z \leftarrow w, K), I \sqsubseteq K \quad (11)$$

$$clause(x \bigcap y, (z \leftarrow u, v), I) \leftarrow clause(x, z \leftarrow u, K), clause(y, z \leftarrow v, J), I \sqsubseteq K \sqcap J \quad (12)$$

A clause $x \leftarrow y \,\square K$ of a plain program P is now represented at the meta-level by

$$clause(P, x \leftarrow y, I) \leftarrow I \sqsubseteq K \quad (13)$$

Note that the condition *non-empty(I)* is not needed in clauses (8)–(13) since it is subsumed by the fact that $I \sqsubseteq H$ for some H.

4. Abstract semantics

Meta-logic provides a semantics to program expressions extended with time intervals in the sense that its axioms tell us how to compute goals of the form

$$demo(\mathscr{E}, G, I)$$

The meta-logical axioms define a new SLD procedure, capable of handling program expressions and time intervals, in terms of the basic SLD of logic programming. An important question is whether it is possible to give a declarative semantics to program expressions. The results reported in Brogi and Turini (1994) show, however, that the plain minimal Herbrand model semantics of logic programs do not lift smoothly to program expressions. Fundamental properties of semantics, like compositionality and full abstraction, are definitely lost. Intuitively speaking, the semantics of program expressions intimately depends on the structure of the plain programs involved, while the minimal Herbrand model semantics forgets it completely. The way around this problem is to use a higher order semantics. The semantics of a program expression is assumed to be the immediate consequence operator associated to it, i.e. a function from Herbrand interpretations to Herbrand interpretations. The immediate consequence operator of a program expression can be compositionally defined in terms of the immediate consequence operator of the sub-expressions. In Brogi and Turini (1994) full abstractness properties of this semantics are also shown. The minimal model semantics can be computed by taking the least fixed point of the immediate consequence operator associated to a program expression.

Here we extend the immediate consequence operator of program expressions to time intervals. In order to do that we consider extended interpretations made up of atom-interval pairs. The immediate consequence operator $\mathscr{T}$ is a function:

$$\mathscr{T} : Exp \longrightarrow \wp(\mathscr{B} \times Int) \longrightarrow \wp(\mathscr{B} \times Int),$$

where $\mathscr{B}$ denotes the Herbrand base, $\wp$ the power-set constructor and Int the set of closed or right unlimited intervals of natural numbers, i.e.

$$Int = \{A \in \wp(\mathbb{N}) \mid (\exists a, b \in \mathbb{N}.\ a \leqslant b \ \wedge\ \forall x.\ x \in A \iff a \leqslant x \leqslant b) \vee (\exists a \in \mathbb{N}.\ \forall x.\ x \in A \iff a \leqslant x)\}.$$

In the following, elements of Int will be simply called *intervals*. It is worth noting that an interval is never empty and that the intersection of intervals is an interval if and only if it is not empty.

Given a program expression $\mathscr{E}$, we write $\mathscr{F}_{\mathscr{E}}$ to denote $\mathscr{F}(\mathscr{E})$. The definition of $\mathscr{F}_{\mathscr{E}}$ is given by cases, according to the structure of $\mathscr{E}$.

- ($\mathscr{E}$ is a plain program P)

$$\mathscr{F}_P(\mathscr{I}) = \left\{ (A, I) \mid \begin{array}{l} \exists (B_1, I_1), \ldots, (B_n, I_n) \in \mathscr{I} \ \wedge \ A \leftarrow B_1, \ldots, B_n \Box I_0 \in ground(P) \ \wedge \\ I = \bigcap_{j=0,n} I_j \ \wedge \ I \neq \varnothing \end{array} \right\}$$

  where $ground(P)$ denotes the set of ground instances of clauses of P.

- ($\mathscr{E} = E_1 \bigcup E_2$)
  $\mathscr{F}_{E_1 \cup E_2}(\mathscr{I}) = \mathscr{F}_{E_1}(\mathscr{I}) \bigcup \mathscr{F}_{E_2}(\mathscr{I})$
- ($\mathscr{E} = E_1 \bigcap E_2$)
  $\mathscr{F}_{E_1 \cap E_2}(\mathscr{I}) = \mathscr{F}_{E_1}(\mathscr{I}) \bigcap' \mathscr{F}_{E_2}(\mathscr{I})$
  where $\mathscr{I}_1 \bigcap' \mathscr{I}_2 = \{(A, I) \mid \exists (A, I_1) \in \mathscr{I}_1, \exists (A, I_2) \in \mathscr{I}_2 : I = I_1 \bigcap I_2 \ \wedge \ I \neq \varnothing\}$

The definition for plain programs differs from the standard logic programming operator just for the conditions on intervals: an atom holds in an interval I, if there exists a clause, holding in a certain interval K, whose head is A, the atoms of the body are solvable and hold in intervals whose intersection with K contains I. The set of immediate consequences of a union of program expressions is the set-theoretic union of the immediate consequences of each program expression. Finally, the set of immediate consequences for the intersection of program expressions consist of atoms, which are consequences of both program expressions and hold in the (non-empty) interval which is the intersection of the intervals associated to the consequences.

To be rigorous, we should use different notations for terms (i.e. pairs of naturals) denoting intervals in the language (syntactic intervals), and sets in the model (semantic intervals), but this would make the notation heavy and cumbersome. Actually, it is always clear from the context whether we are referring to syntax or to semantics.

Moreover, note that the set Int with the operations $\subseteq$ and $\bigcap$ is a model of the theory defining the axiomatization of the order relation between points, inclusion and intersection between syntactic intervals. In other words, as one can trivially prove, for any intervals I, J, K

$$\begin{array}{lll} I \sqsubseteq J \text{ is provable} & \textit{iff} & I \subseteq J \\ J \sqcap K = I & \textit{iff} & J \bigcap K = I. \end{array}$$

It is worth recalling that in the second statement above $I \neq \varnothing$ since I is an interval.

4.1. *Soundness*

We are now in the position of proving the *soundness* of the meta-logical semantics of the program expressions with respect to the abstract semantics. Notice that, since the meta-logical definition axiomatizes a top-down operational semantics for program expressions, the proof corresponds to showing the equivalence of computing program expressions top-down and bottom-up.

In the following we will use $\mathscr{P}$, $\mathscr{Q}$ and $\mathscr{R}$ to denote program expressions, A, B (possibly with subscripts) to denote ground atoms, I, J, K and H (possibly with subscripts) to denote ground intervals and, given a program expression $\mathscr{P}$, $\mathscr{V}$ will

denote the meta-program containing the meta-level representation of the object level programs occurring in $\mathscr{P}$, the axiomatization of the order relation between time points, the subinterval ($\sqsubseteq$) and intersection ($\sqcap$) relations between intervals and the vanilla meta-program consisting of the clauses (7)–(12).

In order to prove the soundness of the meta-interpreter, we first show two lemmas. The first one states that if a conjunctive goal is provable in an interval I, then its atomic conjuncts are provable in intervals whose intersection contains I.

Lemma 1. *Let $\mathscr{P}$ be a program expression and let $\mathscr{V}$ be the corresponding meta-program. For any object level atomic formulas $B_1, \ldots, B_n$ and any ground interval I the following statement holds:*

$$\text{for all } h \quad demo(\mathscr{P}, (B_1, \ldots, B_n), I) \in T_{\mathscr{V}}^h \Longrightarrow \exists I_1, \ldots, I_n : \{demo(\mathscr{P}, B_1, I_1), \ldots, demo(\mathscr{P}, B_n, I_n)\} \subseteq T_{\mathscr{V}}^h \wedge I \subseteq \bigcap_{j=1,n} I_j.$$

Proof. For $n = 1$ the implication trivially holds. For $n \geqslant 2$ the proof can be carried out by induction on h exploiting the definition of $T_{\mathscr{V}}$ and clause (8) of the meta-interpreter. □

The second lemma states that if we can derive a clause $A \leftarrow B_1, \ldots, B_n$ from the meta-program $\mathscr{V}$ and $B_1, \ldots, B_n$ belong to an interpretation $\mathscr{I}$, then the head of the clause, A, is derivable from $\mathscr{P}$ and it holds in an interval containing the intersection of the validity intervals of the clause and of its body.

Lemma 2. (*Virtual clauses lemma*): *Let $\mathscr{P}$ be a program expression and let $\mathscr{V}$ be the corresponding meta-program. For any object level atomic formulas $A, B_1, \ldots, B_n$, any ground intervals $I_0, \ldots, I_n$ and any object level interpretation $\mathscr{I}$, the following statement holds:*

$$clause(\mathscr{P}, (A \leftarrow B_1 \ldots B_n), I_0) \in T_{\mathscr{V}}^{\omega} \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathscr{I} \wedge \left(\bigcap_{j=0,n} I_j\right) \text{ not empty} \Longrightarrow \exists H : (A, H) \in \mathscr{F}_{\mathscr{P}}(\mathscr{I}) \wedge \left(\bigcap_{j=0,n} I_j\right) \subseteq H.$$

Proof. See Appendix.

The soundness of the meta-logical implementation states that an object level atom holding in a certain interval H is derivable from $\mathscr{P}$ if its meta-level representation holding in an interval I contained in H is derivable from $\mathscr{V}$.

Theorem 1. (*Soundness*): *Let $\mathscr{P}$ be a program expression and let $\mathscr{V}$ be the corresponding meta-program. For any object level atomic formula A and any ground interval I, the following statement holds:*

$$demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}^{\omega} \Longrightarrow \exists H : (A, H) \in \mathscr{F}_{\mathscr{P}}^{\omega} \wedge I \subseteq H.$$

Proof. We first show that for all p

$$demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}^{p} \Longrightarrow \exists H : (A, H) \in \mathscr{F}_{\mathscr{P}}^{\omega} \wedge I \subseteq H. \qquad (14)$$

The proof is by induction on p.

(Base case). Trivial since $T_{\mathscr{V}}^0 = \varnothing$.

(Inductive case). Assume that $demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}^{p} \Longrightarrow \exists H : (A, H) \in \mathscr{F}_{\mathscr{P}}^{\omega} \wedge I \subseteq H$.

Then:

$$
\begin{aligned}
& demo(\mathscr{P}, A, I) \in T_{\gamma}^{p+1} \\
\Longleftrightarrow \quad & \{\text{definition of } T_{\gamma}^{\alpha}\} \\
& demo(\mathscr{P}, A, I) \in T_{\gamma}(T_{\gamma}^{p}) \\
\Longleftrightarrow \quad & \{\text{definition of } T_{\gamma} \text{ and clause (9)}\} \\
& \exists K, J: (demo(\mathscr{P}, A, I) \leftarrow clause(\mathscr{P}, A \leftarrow G, K), \\
& demo(\mathscr{P}, G, J), I \sqsubseteq K \sqcap J) \in ground(\mathscr{V}) \wedge \\
& \{clause(\mathscr{P}, A \leftarrow G, K), demo(\mathscr{P}, G, J)\} \subseteq T_{\gamma}^{p} \wedge I \subseteq K \cap J \\
\Longrightarrow \quad & \{\text{Lemma 1 and } G = B_1, \ldots, B_n\} \\
& \exists I_1, \ldots, I_n: \{clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), K), demo(\mathscr{P}, B_1, I_1), \ldots, \\
& demo(\mathscr{P}, B_n, I_n)\} \subseteq T_{\gamma}^{p} \wedge J \subseteq \bigcap_{j=1,n} I_j \wedge I \subseteq K \cap J \\
\Longrightarrow \quad & \{\text{inductive hypothesis}\} \\
& \exists H_1, \ldots, H_n: clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), K) \in T_{\gamma}^{p} \wedge \{(B_1, H_1), \ldots, \\
& (B_n, H_n)\} \subseteq \mathscr{F}_{\mathscr{P}}^{\omega} \wedge I_1 \subseteq H_1 \wedge \ldots \wedge I_n \subseteq H_n \wedge J \subseteq \bigcap_{j=1,n} I_j \wedge I \subseteq K \cap J \\
\Longrightarrow \quad & \{\text{monotonicity of } T_{\gamma} \text{ and } I \subseteq K \cap J \text{ and } J \subseteq \bigcap_{j=1,n} I_j\} \\
& clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), K) \in T_{\gamma}^{\omega} \wedge \{(B_1, H_1), \ldots, (B_n, H_n)\} \subseteq \mathscr{F}_{\mathscr{P}}^{\omega} \wedge \\
& I_1 \subseteq H_1 \wedge \ldots \wedge I_n \subseteq H_n \wedge I \subseteq K \cap \left(\bigcap_{j=1,n} I_j\right) \\
\Longrightarrow \quad & \{\text{Lemma 2 and } K \cap \left(\bigcap_{j=1,n} H_j\right) \text{ is not empty because } K \cap \left(\bigcap_{j=1,n} I_j\right) \\
& \subseteq K \cap \left(\bigcap_{j=1,n} H_j\right) \text{ and } K \cap \left(\bigcap_{j=1,n} I_j\right) \text{ is not empty since it contains } I\} \\
& \exists H: (A, H) \in \mathscr{F}_{\mathscr{P}}(\mathscr{F}_{\mathscr{P}}^{\omega}) \wedge (K \cap \left(\bigcap_{j=1,n} H_j\right)) \subseteq H \wedge \\
& I_1 \subseteq H_1 \wedge \ldots \wedge I_n \subseteq H_n \wedge I \subseteq K \cap \left(\bigcap_{j=1,n} I_j\right) \\
\Longrightarrow \quad & \{\mathscr{F}_{\mathscr{P}}^{\omega} \text{ is a fixpoint of } \mathscr{F}_{\mathscr{P}} \text{ and } K \cap \left(\bigcap_{j=1,n} I_j\right) \subseteq K \cap \left(\bigcap_{j=1,n} H_j\right)\} \\
& \exists H: (A, H) \in \mathscr{F}_{\mathscr{P}}^{\omega} \wedge I \subseteq H
\end{aligned}
$$

We are now able to prove the soundness of the meta-logical semantics of the program expressions with respect to the abstract semantics.

$$
\begin{aligned}
& demo(\mathscr{P}, A, I) \in T_{\gamma}^{\omega} \\
\Longrightarrow & \{\text{definition of } T_{\gamma}^{\omega}\} \\
& \exists p: demo(\mathscr{P}, A, I) \in T_{\gamma}^{p} \\
\Longrightarrow & \{\text{Statement (14)}\} \\
& \exists H: (A, H) \in \mathscr{F}_{\mathscr{P}}^{\omega} \wedge I \subseteq H
\end{aligned}
$$

□

4.2. *Completeness*

In order to prove the completeness of the meta-interpreter we first prove a lemma stating that if A holds in the interval I in a given interpretation $\mathscr{I}$, we can deduce a clause at the meta-level such that its head is A and the body and the related interval in which it holds belong to $\mathscr{I}$.

Lemma 3. *Let $\mathscr{P}$ be a program expression and let $\mathscr{V}$ be the corresponding meta-program. For any object level atomic formula A, any ground interval I and any object level interpretation $\mathscr{I}$, the following statement holds:*

$$
\begin{aligned}
(A, I) \in \mathscr{F}_{\mathscr{P}}(\mathscr{I}) \Longrightarrow \quad & \exists (B_1, I_1), \ldots, (B_n, I_n), I_0: clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), I_0) \in T_{\gamma}^{\omega} \\
& \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathscr{I} \wedge I = \bigcap_{j=0,n} I_j.
\end{aligned}
$$

Proof. See Appendix.

Now we can prove the completeness of the meta-logical semantics of the program expressions with respect to the abstract semantics.

Theorem 2. (*Completeness*): *Let $\mathscr{P}$ be a program expression and let $\mathscr{V}$ be the corresponding meta-program. For any object level atomic formula A and any ground interval I, the following statement holds*:

$$(A, I) \in \mathscr{F}_{\mathscr{P}}^{\omega} \quad \Longrightarrow \quad demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}^{\omega}.$$

Proof. We first show that for all h

$$(A, I) \in \mathscr{F}_{\mathscr{P}}^{h} \quad \Longrightarrow \quad demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}^{\omega}. \tag{15}$$

The proof is by induction on h.
(Base case). Trivial since $\mathscr{F}_{\mathscr{P}}^{0} = \varnothing$.

(Inductive case). Assume that $(A, I) \in \mathscr{F}_{\mathscr{P}}^{h} \Longrightarrow demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}^{\omega}$

Then:

$(A, I) \in \mathscr{F}_{\mathscr{P}}^{h+1}$
$\Longleftrightarrow$ {definition of $\mathscr{F}_{\mathscr{P}}^{\alpha}$}
$(A, I) \in \mathscr{F}_{\mathscr{P}}(\mathscr{F}_{\mathscr{P}}^{h})$
$\Longrightarrow$ {Lemma 3}
$\exists (B_1, I_1), \ldots, (B_n, I_n), I_0 : \; clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), I_0) \in T_{\mathscr{V}}^{\omega} \wedge$
$\{(B_1, I_1) \ldots, (B_n, I_n)\} \subseteq \mathscr{F}_{\mathscr{P}}^{h} \wedge I = \bigcap_{j=0,n} I_j$
$\Longrightarrow$ {inductive hypothesis}
$clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), I_0) \in T_{\mathscr{V}}^{\omega} \wedge$
$\{demo(\mathscr{P}, B_1, I_1), \ldots, demo(\mathscr{P}, B_n, I_n)\} \subseteq T_{\mathscr{V}}^{\omega} \wedge I = \bigcap_{j=0,n} I_j$
$\Longrightarrow$ {definition of $T_{\mathscr{V}}$ and clause (8) used $n-1$ times and
$T_{\mathscr{V}}^{\omega}$ is a fixpoint of $T_{\mathscr{V}}$ }
$clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), I_0) \in T_{\mathscr{V}}^{\omega} \wedge demo(\mathscr{P}, (B_1, \ldots, B_n), \prod_{j=1,n} I_j) \in T_{\mathscr{V}}^{\omega}$
$\wedge I = \bigcap_{j=0,n} I_j$
$\Longrightarrow$ {definition of $T_{\mathscr{V}}$ and clause (9) and $T_{\mathscr{V}}^{\omega}$ is a fixpoint of $T_{\mathscr{V}}$ }
$demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}^{\omega}$

We now prove the completeness of the meta-logical semantics of the program expressions with respect to the abstract semantics.

$(A, I) \in \mathscr{F}_{\mathscr{P}}^{\omega}$
$\Longrightarrow$ {definition of $\mathscr{F}_{\mathscr{P}}^{\omega}$}
$\exists h : \; (A, I) \in \mathscr{F}_{\mathscr{P}}^{h}$
$\Longrightarrow$ {statement (15)}
$demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}^{\omega}$ □

Theorems 1 and 2 show that a program expression $\mathscr{P}$ and its corresponding meta-program $\mathscr{V}$ have a comparable deductive capability with respect to object level goals. However, the meta-interpreter allows one to prove atoms on smaller intervals than the associated program $\mathscr{P}$ does. For instance, consider the program P consisting of the clause

$$p(a) \leftarrow \Box[0, 10].$$

The abstract semantics is $\mathscr{F}_{P}^{\omega} = \{(p(a), \{0, 1, 2, \ldots, 10\})\}$. On the other hand $T_{\mathscr{V}}^{\omega} \supseteq \{demo(P, p(a), I) \mid I \sqsubseteq [0, 10]\}$.

5. Application to legal reasoning

In this section we show the usefulness of our meta-logic by showing its application to legal reasoning.

The basic idea is that laws and rules are naturally represented in separate theories and that they can be combined in ways that are necessarily more complex than plain merging. Time is another crucial ingredient in the definition of laws and rules. Quite often, rules have to refer to instants of time and, furthermore, they have a validity for a fixed period of time. This is especially true for laws and rules which concern taxation and government budget related regulations in general.

For the sake of clarity, in the following examples we write object programs as named collections of object clauses (instead of using the clumsier meta-level representation), and we use year dates instead of integer time points. Conceptually, the axiomatization of section 3 goes through just by using a more sophisticated successor function. In the sequel, we will use the relations *before* and *after* over dates with the intended semantics, which can be given at the meta-level by extending the *demo* predicate in the obvious way. In the actual implementation, as meta-interpreter in Sicstus Prolog, the operations on intervals are such that the maximal interval that satisfies the operation is computed. For example, $[2, 8] \sqsubseteq [3, x]$ binds x to α, and $[2, x] \sqsubseteq [1, 10]$ binds x to 10.

We start with a classical example in the field of legal reasoning, a small part of the British Nationality Act, where the simple partitioning of the knowledge in separate theories, and the use of the basic union operator allow one to use the temporal information in an orderly way.

The statement

> x obtains the British Nationality at time t
> if x is born in UK at time t and
> t is after commencement and
> y is parent of x and
> y is a British citizen at time t
> or y is a British resident at time t

is translated into the following theory.

BNA:

get-citizenship(x,t) ←
born(x,UK,t), after(t,<Jan 1 1955>), parent(y,x), british-citizen(y,t) □
[<Jan 1 1955>,α]

get-citizenship(x,t) ←
born(x,UK,t), after(t,<Jan 1 1955>), parent(y,x), british-resident(y,t) □
[<Jan 1 1955>,α]

Now, we can encode in a separate theory the data proper of a specific individual, say John.

John:

born(John,UK,<Aug 10 1969>) ←□
[<Aug 10 1969>, α]

parent(Bob,John) ←□
[<Aug 10 1969>, α]

british-citizen(Bob,t) ←
after(t,<Sept 6 1940>) □
[<Sept 6 1940>, α]

We can now use the union operator to inquiry about the citizenship of John,

$$\leftarrow demo(BNA \cup John,\ get\text{-}citizenship(John,t),\ -)$$

obtaining as result t = <Aug 10 1969>.

The use of the intersection operator provides a natural way of imposing constraints on existing theories. In general, suppose a theory is given that establishes the validity of a certain property, by means of clauses of the form

property(x,y,z) ← Body □ [a,b].

By intersecting the above theory with a theory containing a clause of the form

property(x,y,z) ← Body′ □ [c,d].

we constrain the property in two respects. The intersection operator on theories imposes that both *Body* and *Body′* must hold in order to derive the property and that this applies only if the time intervals $[a, b]$ and $[c, d]$ do overlap. Intuitively, this corresponds to a new object level rule

property(x,y,z) ← Body, Body′ □ I

where $I \sqsubseteq [a,b] \sqcap [c,d]$.

In the following examples we show the above points. The first example concerns a body of Italian regulations dealing with paying taxes on real estate transactions, called Invim. The original regulation depends on time calculations, since the amount of taxes depends on the period of ownership of the real estate property. Furthermore, the law was abolished in 1992, that means that the rules still apply but only for the period antecedent to 1992.

Our approach allows us to have a theory containing the original regulation and to have two other theories, one containing the constraints due to the decisions taken in 1992, and the other containing the new policy. It is important to notice that the design of the constraining theory can be done without taking care of the details (which may be quite complicated) embodied in the original law.

The following theory—*Invim*—contains a sketch of the original body of regulations.

Invim:

due(amount,x,property) ←
  buys(x,property,t1),
  sells(x,property,t2),
  compute(amount,x,property,t1,t2) □
  [<Jan 1 1950>,α]

compute(amount,x,property,t1,t2) ←. . . □
  [<Jan 1 1950>,α]

In order to adapt the above body of regulations to the new situation imposed by the 1992 decisions, we construct two new theories. The first one is designed as a set of constraints on the applicability of the original rules, while the second one is designed to embody new rules capable of handling the new situation.

Constraints:

due(amount,x,property) ←
  sells(x,property,t),
  before(t,<Dec 31 1992>) □
  [<Jan 1 1993>,α]

compute(amount,x,property,t1,t2) ←□
  [<Jan 1 1993>,α]

The first rule specifies that the relation *due* is computed, i.e. its original body is computed, provided that the selling date is antecedent to 31 December 1992. The second rule specifies that the rules for compute, whatever number they are, and whatever complexity they have, carry on unconstrained to the new version of the regulation.

Additions:

due(amount,x,property) ←
buys(x,property,t1),
sells(x,property,t2),
before(t1,<Dec 31 1992>),
after(t2,<Jan 1 1993>),
compute(amount,x,property,t1,<Dec 31 1992>) □
[<Jan 1 1993>,α]

This rule handles the case of selling a property, bought before 31 December 1992, after 1 January 1993.

Now, we consider a separate theory representing the transactions regarding a specific individual, say Mary, who bought an apartment on 8 March 1965 and sold it on 2 July 1997.

Trans1:

buys(Mary,Apt8,<Mar 8 1965>) ←□ [<Mar 8 1965>,α] sells(Mary,Apt8,<Jul 2 1997>) ←□ [<Jul 2 1997>,α]

The query

←*demo*(*Invim* ⋃ *Trans1, due*(*amount,Mary,Apt8*),–)

yields the amount, say 32.1, that Mary has to pay when selling the apartment according to the old regulations. On the other hand, the query

←*demo*(((*Invim* ⋂ *Constraints*) ⋃ *Additions*) ⋃ *Trans1, due*(*amount,Mary,Apt8*),–)

yields the amount, say 27.8, according to the new regulations.

In the following transaction Paul buys the flat on 1 January 1995.

Trans2:

buys(Paul,Apt9,<Jan 1 1995>) ←□ [<Jan 1 1995>,α] sells(Paul,Apt9,<Sep 12 1996>) ←□ [<Sep 12 1996>,α]

←*demo*(*Invim* ⋃ *Trans2, due*(*amount,Paul,Apt9*),–)

amount = 1.7

←*demo*(((*Invim* ⋂ *Constraints*) ⋃ *Additions*) ⋃ *Trans2, due*(*amount,Paul,Apt9*), –)

no

If we query the theory *Invim* ⋃ *Trans2* Paul must pay a certain amount of tax, say 1.7, but if we consider the updated regulation he must not pay the Invim tax because he bought and sold the flat after 31 December 1992. This is why the answer to the query with respect to the theory ((*Invim* ⋂ *Constraints*) ⋃ *Additions*) ⋃ *Trans2* is *no*.

Trans3:

buys(Frank,Apt10,<May 12 1970>) ←□ sells(Frank,Apt10,<Oct 14 1991>) ←□
[<May 12 1970>,α] [<Oct 14 1991>,α]

←*demo*(*Invim* ⋃ *Trans3, due*(*amount,Frank,Apt10*), –)

amount = 21.3

←*demo*(((*Invim* ⋂ *Constraints*) ⋃ *Additions*) ⋃ *Trans3, due*(*amount,Frank,Apt10*), –)

amount = 21.3

In transaction 3 the computed answers are the same for both queries because the flat has been sold before the abolition of the Invim tax.

The third example deals with *buy & sell subsidized houses*. A person can buy a subsidized house if this house has been assigned to him or he lives with a person who has this house and he agrees to sell it. After 7 July 1993 to buy a subsidized house, it is necessary that the buyer lives in the house for more than five years.

The theory *Legal-buyer* contains the features of the person who can buy a house.

Legal-buyer:

can-buy(x,home) ←
assignee(x,home) □
[<Jan 10 1980>,α]

can-buy(x,home) ←
assignee(y,home),
lives-with(x,y,home),
agrees(x,y,home) □
[<Jan 10 1980>,α]

The theory *Constraints* contains the further constraints imposed after 7 July 1993.

Constraints:

can-buy(x,home) ←
filed-request(x,home,t1),
lives-in-since(x,home,t2),
diffyears(t1,t2,delta),
greaterthan(delta, 5) □
[<Jul 7 1993>,α]

Now, we consider a person who wants to buy a subsidized house.

Marco:

assignee(Luigi,H26) ←□
[<Jan 1 1987>,α]

lives-with(Marco,Luigi,H26) ←□
[<Jan 8 1988>,α]

agrees(Marco,Luigi,H26) ←□
[<Apr 3 1995>,α]

lives-in-since(Marco,H26,<Jan 8 1988>) ←□
[<Jan 8 1988>,α]

The theory *Appl283* is formed by a single clause that records the request.

Appl283:

filed-request(Marco,H26, <Feb 2 1995>) ←□
[<Feb 2 1995>,α]

We can inquire of the system whether Marco can buy the house $H26$ in the following way

$\leftarrow demo(((Legal\text{-}buyer \cap Constraints) \cup Appl283) \cup Marco,\ can\text{-}buy(Marco,H26),i)$

$i = [<\text{Apr 3 1995}>,\alpha]$

From 3 April 1995 Marco can legally buy the house $H26$.

6. Derived operators on time intervals

In order to make easier the selection of information holding in a certain interval we introduce the following operator.

Definition 1. *Let P be a program and I a ground interval.*

$$P \downarrow I = P \cap 1_P^I,$$

where 1_P^I is a theory defined as follows:

for all p defined in P with arity n

$$p(x_1, \ldots, x_n) \leftarrow \Box I.$$

This operator exploits the features of the intersection operator already pointed out in the previous section. What we do is to select only the clauses belonging to P that hold in I or in a subinterval of I and we restrict their validity time to such an interval.

Consider the example of the British Nationality, we want to know whether John was a British citizen on 10 September 1973. The desired query is

$\leftarrow demo((BNA \cup John) \downarrow [<\text{Sept 10 1973}>,<\text{Sept 10 1973}>], \text{get-citizenship(John,-),-}).$

The answer is *yes* because John got his citizenship on 10 August 1969. An equivalent way to express this query is

$\leftarrow demo((BNA \cup John), \text{get-citizenship(John,-)},[<\text{Sept 10 1973}>,<\text{Sept 10 1973}>]),$

where we temporally constrain the validity time of the goal itself rather than the theory of the goal. The advantage of having the operator $P \downarrow I$ is that we can ask for query of the kind $A \in P \downarrow t_1 \cup P \downarrow t_2$ in order to join, for instance, information derived in different dates.

Another useful application of the intersection operator consists in modelling updates. Suppose that we want to represent the fact that Frank is a researcher in mathematics, then he is promoted and becomes an assistant professor. To model information that can be updated we use unit clauses. In our formalism we define a theory *Frank* that records the information associated to Frank as researcher.

Frank:

researcher(maths) $\Box$
$[<\text{Mar 8 1993}>,\alpha]$

In March 1996 Frank became an assistant professor. In order to modify the information contained in the theory *Frank*, we build this new theory:

$$(Frank \cap \{\text{researcher(maths)} \Box [-, < 29\ \text{Feb}\ 1996 >] \}) \cup$$
$$\{\text{assistant\_professor(maths)} \Box [< 1\ \text{Mar}\ 1996 >, \alpha]\}$$

We have used an unnamed theory that is represented by the following meta-level clause.

$$demo(\{X \leftarrow Y \Box I\}, X \leftarrow Y, K) \leftarrow K \sqsubseteq I$$

This update is similar to add and delete a ground atom. For instance in $\mathscr{LPS}$ + + (Zaniolo et al. 1993) we can express such a change by solving the goal $\neg researcher(maths), + assistant\_professor(maths)$.

The advantage of our approach is that we do not change directly the clauses of the theory *Frank* but we compose the old theory with a new one that represents the current situation. Therefore we preserve even the state of the database before 1 March 1996, keeping faith to the historical dealing of information.

$\leftarrow demo((Frank \cap \{researcher(maths) \Box [-, < 29\ Feb\ 1996 >] \}) \cup$
$\{assistant\_professor(maths) \Box [< 1\ Mar\ 1996 >, \alpha]\},$
$researcher(X), [< 23\ Feb\ 1994 >, < 23\ Feb\ 1994 >])$
X = maths

$\leftarrow demo((Frank \cap \{researcher(maths) \Box [-, < 29\ Feb\ 1996 >] \}) \cup$
$\{assistant\_professor(maths) \Box [< 1\ Mar\ 1996 >, \alpha]\},$
$researcher(X), [< 12\ Mar\ 1996 >, < 12\ Mar\ 1996 >])$
no.

The first query inquires the updated database before the advance in career of Frank, conversely, the second shows how information in the database has been modified.

7. Reasoning over joined intervals

The meta-interpreter of section 3 does not allow us, given a program expression, to compute a maximal interval in which a query holds. For example, consider the following theories representing two library databases:

DB1:
borrow(Mary,The Twelfth Night) $\leftarrow \Box$
[<May 12 1995>,<Jun 12 1995>]

DB2:
borrow(Mary,The Twelfth Night) $\leftarrow \Box$
[<Jun 12 1995>, <August 1 1995>]

By querying the union of the above theories we would obtain the period of time in which Mary has borrowed The Twelfth Night,

$$\leftarrow demo(DB1 \cup DB2, \text{borrow(Mary,The Twelfth Night)}, i).$$

According to the semantics considered so far, the above query computes two answers:

i = [<May 12 1995>,<Jun 12 1995>] i = [<Jun 12 1995>,<August 1 1995>]

Actually, Mary has borrowed the book from 12 May 1995 until 1 August 1995 which can be obtained by joining the previous answers and we would like to obtain also such an interval as a computed answer substitution.

Reasoning over joined intervals requires first to axiomatize the *join* of intervals, denoted by $\sqcup$.

$$[a,b] \sqcup [c,d] = [a,d] \Leftarrow non\text{-}empty([a,b]), non\text{-}empty([c,d]), a \leqslant c, c \leqslant b+1, b \leqslant d$$
$$[a,b] \sqcup [c,d] = [a,b] \Leftarrow non\text{-}empty([a,b]), non\text{-}empty([c,d]), a \leqslant c, d < b$$
$$[a,b] \sqcup [c,d] = [c,b] \Leftarrow non\text{-}empty([a,b]), non\text{-}empty([c,d]), c \leqslant a, a \leqslant d+1, d \leqslant b$$
$$[a,b] \sqcup [c,d] = [c,d] \Leftarrow non\text{-}empty([a,b]), non\text{-}empty([c,d]), c \leqslant a, b < d$$

It is worth noting that join is defined on *overlapping intervals* or on *meeting intervals* that is $[a,b]$ and $[b+1,c]$ and it is commutative. Unfortunately, join of intervals is not associative. For instance, consider the intervals $I_1 = [1,5]$, $I_2 = [2,3]$ and $I_3 = [6,7]$, then $(I_1 \sqcup I_2) \sqcup I_3 = [1,7]$ but $I_1 \sqcup (I_2 \sqcup I_3)$ does not exist because I_2 and I_3 are neither overlapping nor meeting intervals. We write $\bigsqcup_{j=1,n} I_j$ to denote $((I_1 \sqcup I_2) \ldots \sqcup I_n)$.

Set-theoretic union restricted to Int is the semantic counterpart of $\sqcup$, i.e. for any intervals I, J, K

$$K \sqcup J = I \text{ is provable } \textit{iff } K \bigcup J = I.$$

Note that union of intervals, although it is surely non-empty as a set, is not necessarily an interval since the original intervals may be neither overlapping nor meeting.

However, when joining more than two intervals, the lack of associativity for $\sqcup$ breaks the correspondence with set-theoretic union $\bigcup$. In fact it is still true that if $((I_1 \sqcup I_2) \sqcup I_3) = K$ then $\bigcup_{j=1,3} I_j = K$ but the converse in general does not hold. Anyway, it is not difficult to prove that if the union of $I_1, \ldots, I_n$ is an interval K then joining $I_1, \ldots, I_n$ in a suitable order, we can exactly obtain K as the result, i.e.

$$\textit{if } \bigcup_{j=1,n} I_j = K \wedge K \in Int \textit{ then } \exists \textit{ an ordering } i_1, \ldots, i_n \textit{ of } 1, \ldots, n$$
$$\textit{such that } ((I_{i_1} \sqcup I_{i_2}) \ldots \sqcup I_{i_n}) = K.$$

As an easy consequence, if $(I_1 \sqcup I_2) \sqcup I_3$ and $I_1 \sqcup (I_2 \sqcup I_3)$ are defined then they coincide.

Finally in the sequel we will use the following simple property.

Proposition 3. *Let $I_1, \ldots, I_n, H_1, \ldots, H_n$ be intervals such that $I_j \subseteq H_j$ for each $j = 1 \ldots n$ and $\bigcup_{j=1,n} I_j$ is an interval. Then $\bigcup_{j=1,n} H_j$ is an interval containing $\bigcup_{j=1,n} I_j$.*

We can now extend the meta-interpreter to take into account joined intervals by adding the following clause:

$$demo(x,y,I) \leftarrow demo(x,y,K), demo(x,y,J), I = K \sqcup J \tag{16}$$

In the previous example, we also obtain i = [<May 12 1995>,<August 1 1995>] as a computed answer substitution to the query

$$\leftarrow demo(DB1 \bigcup DB2, \text{borrow(Mary,The Twelfth Night)}, i).$$

As far as the abstract semantics is concerned, given a program expression $\mathscr{P}$ we define an immediate consequence operator $\mathscr{T}^* : Exp \rightarrow \wp(\mathscr{B} \times Int) \rightarrow \wp(\mathscr{B} \times Int)$

as follows:

$$\mathcal{F}^*_{\mathcal{P}}(\mathcal{I}) = \{(A, I) \mid \exists (A, I_1), \ldots, (A, I_n) \in \mathcal{F}_{\mathcal{P}}(\mathcal{I}) \wedge I = \bigcup_{j=1,n} I_j \wedge I \in Int\}.$$

Again, $\mathcal{F}^*_{\mathcal{P}}$ denotes $\mathcal{F}^*(\mathcal{P})$. The idea is that if an atom A holds in a set of intervals $\{I_1, \ldots, I_n\}$ and $\bigcup_{j=1,n} I_j$ is the interval I then A holds in I.

Now we want to prove that the meta-logical semantics (containing clauses (7)–(12) and (16)) is *sound* and *complete* with respect to the abstract semantics $\mathcal{F}^*$. We follow the line of the proofs of section 4.

Let $\mathcal{P}$ be a program expression, in the following we will denote with $\mathcal{V}^*$ the meta-program containing the meta-level representation of the object level programs occurring in $\mathcal{P}$, the axiomatization of the order relation between time points, the subinterval ($\sqsubseteq$), join ($\sqcup$) and intersection ($\sqcap$) relations between intervals and the vanilla meta-program consisting of the clauses (7)–(12) and (16).

Remark: Let $\mathcal{P}$ be a program expression and let $\mathcal{V}$ and $\mathcal{V}^*$ be the corresponding meta-programs and $\mathcal{I}$ any object level interpretation. The following statements hold:

(i) $\mathcal{F}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{F}^*_{\mathcal{P}}(\mathcal{I})$;
(ii) $T_{\mathcal{V}}(\mathcal{I}) \subseteq \mathcal{T}_{\mathcal{V}^*}(\mathcal{I})$.

Therefore

(iii) $\mathcal{F}^{\omega}_{\mathcal{P}} \subseteq \mathcal{F}^{*\omega}_{\mathcal{P}}$;
(iv) $T^{\omega}_{\mathcal{V}} \subseteq T^{\omega}_{\mathcal{V}^*}$

Lemma 4. *Let $\mathcal{P}$ be a program expression and let $\mathcal{V}$ and $\mathcal{V}^*$ be the corresponding meta-programs. For any object level atomic formulas $A, B_1, \ldots, B_n$ and any ground interval I_0, the following statement holds:*

$$clause(\mathcal{P}, (A \leftarrow B_1, \ldots, B_n), I_0) \in T^{\omega}_{\mathcal{V}} \iff clause(\mathcal{P}, (A \leftarrow B_1, \ldots, B_n), I_0) \in T^{\omega}_{\mathcal{V}^*}.$$

Proof. ($\Longrightarrow$) Trivial for Statement (*iv*) of Remark 1.
($\Longleftarrow$) Trivial by observing that the definitions of the predicate *clause* are not affected by adding clause (16). □

Corollary 1. *Let $\mathcal{P}$ be a program expression and let $\mathcal{V}^*$ be the corresponding meta-program. For any object level atomic formulas $A, B_1, \ldots, B_n$, any ground intervals $I_0, \ldots, I_n$ and any object level interpretation $\mathcal{I}$, the following statement holds:*

$$clause(\mathcal{P}, (A \leftarrow B_1, \ldots, B_n), I_0) \in T^{\omega}_{\mathcal{V}^*} \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathcal{I} \wedge$$
$$\bigcap_{j=0,n} I_j \text{ not empty} \Longrightarrow \exists H : (A, H) \in \mathcal{F}^*_{\mathcal{P}}(\mathcal{I}) \wedge \bigcap_{j=0,n} I_j \subseteq H$$

Proof. Immediate from Lemma 4, Lemma 2 and Statement (*i*) of Remark 1. □

We can reformulate Theorems 1 and 2 in order to prove the equivalence between the extended meta-interpreter and the above definition of $\mathcal{F}^*_{\mathcal{P}}$. First we prove a generalization of Lemma 1 about conjunctive goals. If a conjunctive goal, composed by n atoms and holding in an interval I, is derivable from the meta-interpreter $\mathcal{V}^*$, then for each conjunct there exists a set of intervals such that their union is an interval, the conjunct holds in each such interval and is derivable from $\mathcal{V}$, i.e. from the meta-interpreter without clause (16). Moreover, the intersection of the union of such sets of intervals is not empty and it is an interval.

Lemma 5. *Let $\mathscr{P}$ be a program expression and let $\mathscr{V}$ and $\mathscr{V}^*$ be the corresponding meta-programs. For any object level atomic formulas $B_1, \ldots, B_n$ and any ground interval I, the following statement holds:*

$$\begin{aligned}\text{for all } h\ demo(\mathscr{P}, (B_1, \ldots, B_n), I) \in T_{\mathscr{V}}{}^{h}{}_{*} \implies \exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^n, \ldots, I_{i_n}^n, q :\\ \{demo(\mathscr{P}, B_j, I_r^j) \mid j = 1..n, r = 1..i_j\} \subseteq T_{\mathscr{V}}{}^{q} \wedge\\ I \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} I_j^n\right) \wedge\\ \bigcup_{r=1,i_j} I_r^j \in Int \text{ for each } j = 1..n.\end{aligned}$$

Proof. See Appendix.

We are now ready to prove the soundness and completeness of the meta-interpreter extended to deal with joined intervals.

Theorem 4. (*Soundness*): *Let $\mathscr{P}$ be a program expression and let $\mathscr{V}^*$ be the meta-program. For any object level atomic formula A and any ground interval I, the following statement holds:*

$$demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}{}^{\omega}{}_{*} \implies \exists H : (A, H) \in \mathscr{F}_{\mathscr{P}}^{*\,\omega} \wedge I \subseteq H.$$

Proof.

$$\begin{aligned}&demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}{}^{\omega}{}_{*}\\ \implies\ &\{\text{definition of } T_{\mathscr{V}}{}^{\omega}{}_{*}\}\\ &\exists p :\ demo(\mathscr{P}, A, I) \in T_{\mathscr{V}}{}_{*}(T_{\mathscr{V}}{}^{p}{}_{*})\\ \iff\ &\{\text{definition of } T_{\mathscr{V}}{}_{*}\}\\ &\exists Body : (demo(\mathscr{P}, A, I) \leftarrow Body) \in ground(\mathscr{V}^*) \wedge Body \subseteq T_{\mathscr{V}}{}^{p}{}_{*}\end{aligned}$$

We have two cases corresponding to clauses (9) and (16):

(*a*) $Body = clause(\mathscr{P}, A \leftarrow G, K), demo(\mathscr{P}, G, J), I \sqsubseteq K \sqcap J$;
(*b*) $Body = demo(\mathscr{P}, A, K), demo(\mathscr{P}, A, J), I = K \sqcup J$

$$\begin{aligned}(a)\ &\exists K, J : (demo(\mathscr{P}, A, I) \leftarrow clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), K), demo(\mathscr{P}, (B_1, \ldots, B_n), J),\\ &I \sqsubseteq K \sqcap J) \in ground(\mathscr{V}^*) \wedge \{clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), K),\\ &demo(\mathscr{P}, (B_1, \ldots, B_n), J)\} \subseteq T_{\mathscr{V}}{}^{p}{}_{*} \wedge I \subseteq K \cap J\\ \implies\ &\{\text{Lemma 5}\}\\ &\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^n, \ldots, I_{i_n}^n, q :\ \{demo(\mathscr{P}, B_j, I_r^j) \mid j = 1..n, r = 1..i_j\} \subseteq T_{\mathscr{V}}{}^{q} \wedge\\ &J \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} I_j^n\right) \wedge \bigcup_{r=1,i_j} I_r^j \in Int \text{ for each } j = 1..n \wedge\\ &clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), K) \in T_{\mathscr{V}}{}^{p}{}_{*} \wedge I \subseteq K \cap J\\ \implies\ &\{\text{Statement (14) of Theorem 1}\}\\ &\exists H_1^1, \ldots, H_{i_1}^1, \ldots, H_1^n, \ldots, H_{i_n}^n :\\ &\{(B_j, H_r^j) \mid j = 1..n, r = 1..i_j\} \subseteq \mathscr{F}_{\mathscr{P}}^{\omega} \wedge I_1^1 \subseteq H_1^1, \ldots, I_{i_n}^n \subseteq H_{i_n}^n \wedge\\ &J \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} I_j^n\right) \wedge \bigcup_{r=1,i_j} I_r^j \in Int \text{ for each } j = 1..n \wedge\\ &clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), K) \in T_{\mathscr{V}}{}^{p}{}_{*} \wedge I \subseteq K \cap J\\ \implies\ &\left\{\left(\bigcup_{r=1,i_j} I_r^j\right) \subseteq \left(\bigcup_{r=1,i_j} H_r^j\right) \text{ and } \bigcup_{r=1,i_j} H_r^j \in Int \text{ for each } j \in \{1, \ldots, n\}\right.\\ &\left.\text{by Proposition 3 and definition of } \mathscr{F}_{\mathscr{P}}^{*}\right\}\\ &\exists H_1^1, \ldots, H_{i_1}^1, \ldots, H_1^n, \ldots, H_{i_n}^n :\ \{(B_j, \bigcup_{r=1,i_j} H_r^j) \mid j = 1..n\} \subseteq \mathscr{F}_{\mathscr{P}}^{*}(\mathscr{F}_{\mathscr{P}}^{\omega}) \wedge\\ &J \subseteq \left(\bigcup_{j=1,i_1} H_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} H_j^n\right) \wedge \bigcup_{r=1,i_j} H_r^j \in Int \text{ for each } j = 1..n \wedge\\ &clause(\mathscr{P}, (A \leftarrow B_1, \ldots, B_n), K) \in T_{\mathscr{V}}{}^{p}{}_{*} \wedge I \subseteq K \cap J\\ \implies\ &\{\text{Corollary 1 and } K \cap \left(\bigcap_{j=1,n} \left(\bigcup_{s=1,i_j} H_s^j\right)\right)\end{aligned}$$

$$
\begin{array}{ll}
 & \text{is not empty since } I \subseteq K \cap J \text{ and } I \neq \emptyset\} \\
 & \exists H:\ (A,H) \in \mathcal{F}^{*}_{\mathcal{P}}(\mathcal{F}^{*}_{\mathcal{P}}(\mathcal{F}^{\omega}_{\mathcal{P}})) \ \wedge\ K \cap \left(\bigcap_{j=1,n}\left(\bigcup_{s=1,i_j} H^{j}_{s}\right)\right) \subseteq H \ \wedge \\
 & J \subseteq \left(\bigcup_{j=1,i_1} H^{1}_{j}\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} H^{n}_{j}\right) \ \wedge\ I \subseteq K \cap J \\
\Longrightarrow & \{I \subseteq K \cap J, \text{ Statement (iii) of Remark 1 and } \mathcal{F}^{*\,\omega}_{\mathcal{P}} \text{ is a fixpoint of } \mathcal{F}^{*}_{\mathcal{P}}\} \\
 & \exists H:\ (A,H) \in \mathcal{F}^{*\,\omega}_{\mathcal{P}} \ \wedge\ I \subseteq H
\end{array}
$$

$$
\begin{array}{ll}
(b)\ \exists K,J:\ (demo(\mathcal{P},A,I) \leftarrow demo(\mathcal{P},A,K), demo(\mathcal{P},A,J), I = K \sqcup J) \in \\
\quad ground(\mathcal{V}^{*}) \ \wedge \{demo(\mathcal{P},A,K), demo(\mathcal{P},A,J)\} \subseteq T^{p}_{\mathcal{V}*} \ \wedge\ I = K \cup J \ \wedge\ I \in Int \\
\Longrightarrow & \{\text{Lemma 5}\} \\
 & \exists I_1,\ldots,I_n,\overline{I}_1,\ldots,\overline{I}_m,q_1,q_2:\ \{demo(\mathcal{P},A,I_1),\ldots,demo(\mathcal{P},A,I_n)\} \subseteq T^{q_1}_{\mathcal{V}} \ \wedge \\
 & K \subseteq I_1 \cup \ldots \cup I_n \ \wedge\ (I_1 \cup \ldots \cup I_n) \in Int \ \wedge \\
 & \{demo(\mathcal{P},A,\overline{I}_1),\ldots,demo(\mathcal{P},A,\overline{I}_m)\} \subseteq T^{q_2}_{\mathcal{V}} \ \wedge\ J \subseteq (\overline{I}_1 \cup \ldots \cup \overline{I}_m) \ \wedge \\
 & (\overline{I}_1 \cup \ldots \cup \overline{I}_m) \in Int \ \wedge\ I = K \cup J \ \wedge\ I \in Int \\
\Longrightarrow & \{q = \max\{q_1,q_2\}, \text{ monotonicity of } T_{\mathcal{V}} \text{ and Statement (14) of} \\
 & \text{Theorem 1}\} \\
 & \exists H_1,\ldots,H_n,\overline{H}_1,\ldots,\overline{H}_m:\ \{(A,H_1),\ldots,(A,H_n),(A,\overline{H}_1),\ldots,(A,\overline{H}_m)\} \subseteq \mathcal{F}^{\omega}_{\mathcal{P}} \ \wedge \\
 & I_1 \subseteq H_1,\ldots,I_m \subseteq H_m \ \wedge\ K \subseteq (I_1 \cup \ldots \cup I_n) \ \wedge\ J \subseteq (\overline{I}_1 \cup \ldots \cup \overline{I}_m) \ \wedge \\
 & (I_1 \cup \ldots \cup I_n) \in Int \ \wedge\ (\overline{I}_1 \cup \ldots \cup \overline{I}_m) \in Int \ \wedge\ I = K \cup J \ \wedge\ I \in Int \\
\Longrightarrow & \{H = (H_1 \cup \ldots \cup H_n) \cup (\overline{H}_1 \cup \ldots \cup \overline{H}_m), \text{ and } H \in Int \text{ by Proposition 3,} \\
 & \text{Statement (iii) of Remark 1, definition of } \mathcal{F}^{*}_{\mathcal{P}} \text{ and } \mathcal{F}^{*\,\omega}_{\mathcal{P}} \\
 & \text{is a fixpoint of } \mathcal{F}^{*}_{\mathcal{P}}\} \\
 & \exists H:\ (A,H) \in \mathcal{F}^{*\,\omega}_{\mathcal{P}} \ \wedge\ I \subseteq H
\end{array}
$$

□

Theorem 5. (*Completeness*): *Let $\mathcal{P}$ be a program expression and let $\mathcal{V}^{*}$ be the metaprogram. For any object level atomic formula A and any ground interval I, the following statement holds:*

$$(A,I) \in \mathcal{F}^{*\,\omega}_{\mathcal{P}} \quad \Longrightarrow \quad demo(\mathcal{P},A,I) \in T^{\omega}_{\mathcal{V}*}$$

Proof.

$$
\begin{array}{ll}
 & (A,I) \in \mathcal{F}^{*\,\omega}_{\mathcal{P}} \\
\Longleftrightarrow & \{\text{definition of } \mathcal{F}^{*\,\omega}_{\mathcal{P}}\} \\
 & \exists h:\ (A,I) \in \mathcal{F}^{*\,h}_{\mathcal{P}} \\
\Longleftrightarrow & \{\text{definition of } \mathcal{F}^{*}_{\mathcal{P}}\} \\
 & \exists I_1,\ldots,I_n:\ \{(A,I_1),\ldots,(A,I_n)\} \subseteq \mathcal{F}^{h}_{\mathcal{P}} \ \wedge\ I = \bigcup_{j=1,n} I_j \ \wedge\ I \in Int \\
\Longrightarrow & \{\text{Statement (15) of Theorem 2}\} \\
 & \{demo(\mathcal{P},A,I_1),\ldots,demo(\mathcal{P},A,I_n)\} \subseteq T^{\omega}_{\mathcal{V}} \ \wedge\ I = \bigcup_{i=1,n} I_j \ \wedge\ I \in Int \\
\Longrightarrow & \{\text{Statement (iv) of Remark 1}\} \\
 & \{demo(\mathcal{P},A,I_1),\ldots,demo(\mathcal{P},A,I_n)\} \subseteq T^{\omega}_{\mathcal{V}*} \ \wedge\ I = \bigcup_{i=1,n} I_j \ \wedge\ I \in Int \\
\Longrightarrow & \{\text{clause (16) used } n-1 \text{ times in a suitable order and } T^{\omega}_{\mathcal{V}*} \text{ is a} \\
 & \text{fixpoint of } T_{\mathcal{V}*}\} \\
 & demo(\mathcal{P},A,I) \in T^{\omega}_{\mathcal{V}*}
\end{array}
$$

□

8. Related work

Interest in research concerning the handling of temporal information has been growing steadily over the past two decades. On the one hand, a lot of effort has been spent in developing extensions of logical languages capable to deal with time (see, for example, Orgun and Ma (1994)). On the other hand, in the field

of databases, many approaches have been proposed to extend the existing data models, such as the *relational*, the *object-oriented* and the *deductive* models, to cope with temporal data (see, e.g. the book by Tansel et al. (1993)). It is clear that a close connection exists between these two trends in research, since temporal logic languages can provide solid theoretical foundations for temporal databases, and powerful knowledge representation and query languages for them (Gabbay and McBrien 1991, Chomicki 1994, Orgun 1996).

Two main logical formalisms have been developed for time: *temporal logic* with modal temporal operators and *classical logic* with temporal variables. In the following we briefly review some approaches in the literature which seem to be relevant to our work, presenting them according to the above classification.

8.1. *Temporal and modal languages*

Several languages based on temporal logic have been defined. The languages such as Chronolog (Rolston 1986, Orgun and Wadge 1988), Templog (Abadi and Manna 1989), Temporal Datalog (Orgun 1996), Temporal Prolog (Gabbay 1987), Temporal Prolog (Sakuragawa 1987), MTL (Brzoska 1995) directly extend logic programming with temporal operators. Most of them use temporal versions of resolution-based proof procedures and they differ in which temporal constructs they allow, and, therefore, in which classes of problems they can naturally express. We discuss only Templog as an example of this class of languages. For a survey of temporal and modal logic programming consider the overview by Orgun and Ma (1994) and for recent trends see Fisher and Owens (1995).

Templog extends classical Horn logic programming languages, such as Prolog (Sterling and Shapiro 1986), to include programs with the following temporal constructs: $\bigcirc u$ (u is true at the next instant of time), $\Box u$ (u is always true (from now on)), $\Diamond u$ (u is eventually true). Programs are sets of temporal clauses subdivided into permanent and initial clauses with particular constraints on the use of the temporal operators in the head and the body of clauses. This approach is point based and it is difficult to ask queries such as *In which period was Frank a researcher?*, that is to extract infomation about duration of actions, intervals in which some facts hold, as we can easily do. This language, as most of the others mentioned, establishes relations among instants of time and for this reason, a disadvantage is that they cannot naturally represent explicit references to time. They are designed to handle relative timing information rather than a quantitative timing one. Templog is based on temporal SLD resolution, a sound and complete proof procedure, that is reasonably efficient. Moreover two equivalent formulations of Templog's declarative semantics are given: in terms of a least fixpoint and in terms of a minimal Herbrand model (Baudinet 1989).

The language MTL, a temporal logic programming language with metric and past operators, presented in Brzoska (1995) subsumes Templog. It allows not only qualitative, but also quantitative temporal information, such as $\Box_{[c_1,c_2]}A$, meaning that *A always holds between* c_1 *and* c_2. An interesting feature of this approach, that we would like to explore even in our framework, is the possibility of translating MTL into an instance of the CLP-scheme (Jaffar and Maher 1994) over a suitable algebra, where (due to the particular form of the involved constraints) the MTL-resolution can be expressed as a quite efficient restriction of CLP-resolution.

Concerning the other mentioned approaches, Gabbay's Temporal Prolog has an

operational semantics that does not immediately fully define an interpreter, because it does not account for the linearity of time and a proof search strategy is not specified. Sakuragawa's Temporal Prolog programs are compiled into Prolog. In our approach we provide a meta-interpreter whose axioms define a new SLD procedure, capable of handling program expressions and time intervals, in terms of the basic SLD of logic programming. Then we give a declarative semantics by using a least fixpoint approach and we prove the equivalence of the meta-logical and least fixpoint semantics.

8.2. *Classical logic languages with temporal variables*

In this category of languages we recall Datalog$_{1S}$ (Chomicki and Imielinski 1988, Baudinet et al. 1993), Temporal Prolog (Hrycej 1993), Temporal Annotated Constraint Logic Programming (Frühwirth 1996), Constraint Databases (Kanellakis et al. 1995, Koubarakis 1994) and Event Calculus (Kowalski and Sergot 1986, Sripada 1988).

Datalog$_{1S}$ is a language for temporal deductive databases (Baudinet *et al.* 1993), which allows for the implicit representation of the extension of the temporal attributes. The main advantage they offer is the gain in expressiveness: they make the representation of infinite extensions possible and they often allow for a more compact representation of finite extensions. Datalog$_{1S}$ is an extension of Datalog such that all the predicates are extended with an extra parameter for time. The time parameter is constructed using a unary function symbol denoting the *successor* function.

Datalog$_{1S}$ shares with our language a fixpoint semantics, a discrete time domain, but differs substantially in the fact that it is point-wise (while our approach is oriented to intervals) and flat (no mechanism for structuring and combining programs is given). A distinctive feature of Datalog$_{1S}$ is the possibility of representing *periodic data*, yet not allowed in our approach. In the next session we will discuss a possible way to overcome this limitation.

Temporal Prolog (Hrycej 1993) extends Prolog with two additional clause types: *temporal references*, denoted by P *in* T, which are used to assert that a certain statement P holds exactly during a time interval T, and *temporal constraints*, which axiomatize Allen's relationships between time intervals. This approach has many features in common with ours: it attaches time intervals to clauses, it uses an interval-based algebra and it gives a meta-level representation of the knowledge via the predicate *Holds*. The main differences reside in the fact that Temporal Prolog does not have any modularization mechanism and it deals with negation and disjunctive formulae that we do not provide. Again we find extremely interesting the way the language has been designed in order to have an efficient implementation based on a temporal constraint solver.

While in Temporal Prolog time is associated with clauses, in Temporal Constraint Annotated Logic Programming (TACLP), presented in Frühwirth (1996), each formula can be labelled (*annotated*) by temporal information, for instance

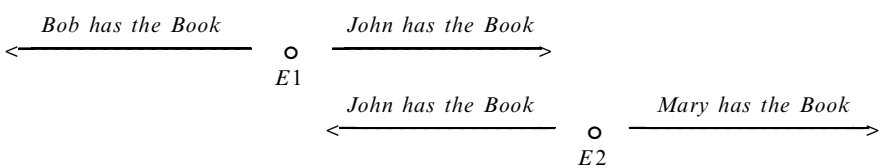
$$owns\_car(X)\ th[T_1, T_2] \leftarrow buys\_car(X)\ at\ T_1 \wedge sells\_car(X)\ at\ T_2$$

means that if a person X buys a car at T_1 and she/he sells it at T_2 then she/he owns the car in the time period from T_1 to T_2. Formulae can have three possible temporal annotations: A *at* T (A holds at time point T), A *th* $[T_1, T_2]$ (A holds throughout the time period from T_1 to T_2) and A *in* $[T_1, T_2]$ (A holds at some point(s) in

$[T_1, T_2]$—indefinite temporal information). Such approach supports both qualitative and quantitative temporal reasoning about definite and indefinite information with time points and time periods. TACLP shares with our approach many features, such as the presence of a meta-interpreter (although implemented in constraint logic programming), the structure of time, the time period order relation (subinterval) and the join operation on intervals defined only on overlapping intervals (in order to always have convex sets). This makes us confident that the treatment of indefinite information, yet not available for our approach, should not be difficult to be imported. The differences pointed out for Hrycej's Temporal Prolog can be repeated even for Temporal annotated logic.

Constraint Databases (Kanellakis *et al.* 1995) generalize the classical relational data model by introducing *generalized tuples*: quantifier-free formulas in an appropriate constraint theory. The key intuition of this approach is that the generalization of a tuple is a conjunction of constraints. By varying the underlying logical theory one can model different problems. In particular, to represent temporal information the *dense linear order inequality theory* is often used as logical theory for constraints. In this theory, for instance, the instant k is modelled by the generalized tuple on variable X, $X = k$ and an interval $[k_1, k_2]$ is modelled as $X \geqslant k_1 \wedge X \leqslant k_2$. We refer to Koubarakis (1994) for a detailed presentation of a significant application of constraint databases to the handling of time. Koubarakis succeeds in modelling not only infinite temporal information but also indefinite information.

Finally we mention Event Calculus (Kowalski and Sergot 1986, Kowalski 1992). Event Calculus is a treatment of time, based on the notion of events, in first-order classical logic augmented with negation as failure. It is closely related to Allen's interval temporal logic (Allen 1984). For example, let E1 be an event in which *Bob gives the Book to John* and E2 be an event in which *John gives Mary the Book*. Assume that E2 occurs after E1. Given these event descriptions, we can deduce that there is a period started by the event E1 in which John possesses the book and that there is a period terminated by E1 in which Bob possesses the book. This situation is represented pictorially



A series of axioms for deducing the existence of time periods and the Start and End of each time period are given by using the *Holds* predicate.

Holds(before(e r)) if Terminates(e r)

means that relationship r holds in the time period *before(e r)* that denotes a time period terminated by the event e. *Holds(after(e r))* is defined in an analogous way. Event Calculus provides a natural treatment of valid time in databases, and it was extended in Sripada (1988, 1991) to include the concept of transaction time.

Therefore Event Calculus exploits the deductive power of logic and the computational power of logic programming as in our approach, but the modelling of time is different: events are the granularity of time chosen in Event Calculus, whereas we use time intervals.

8.3. *Final discussion*

The main difference of our approach from the cited ones, is the context: we are interested in modelling time in a multi-theory framework. Our knowledge (even temporal) is distributed in different theories and we provide operators to query and compose such theories. Almost all the logical languages discussed so far lack these features.

Exceptions are Temporal Datalog (Orgun 1996) and constraint databases (Kanellakis *et al.* 1995). The first one introduces a notion of module. However, this mechanism seems not to be used as a knowledge representation tool but to define new non-standard algebraic operators. In fact, to query a temporal Datalog program, Orgun proposes a 'point-wise extension' of the relational algebra upon the set of natural numbers, called TRA-algebra. Temporal modules can be used to feed back temporal relations, created during the evaluation of TRA expressions, to the deduction part for further manipulation, having access in this way to the use of recursion, arithmetic predicates and temporal operators.

Indeed, our context is closer to the paradigm of constraint databases. Actually, from a deductive database perspective, each logic program of our framework can be viewed as an enriched relational database where relations are represented partly intensionally and partly extensionally. The meta-level operators can then be viewed as a means of constructing views by combining multiple databases in various ways. In reality, if in our theories we had only ground facts, our union $\bigcup$ and intersection $\bigcap$ operators would correspond to the union $\bigcup$ and intersection $\bigcap$ on generalized relations of constraint databases with the dense linear order inequality theory. The database operations we are not able to model are those requiring negation, such as the difference between relations.

But our theories are not just ground facts. We can exploit the deductive power of rules and we have also the possibility of having a dynamic set of rules. In fact, we assign valid time also to clauses and this allows us to change the set of clauses used to solve a goal according to their temporal validity, as we have seen in section 5.

9. Future work

As mentioned in the introduction, we intend to investigate different representations of time, other than the one based on time intervals attached to program clauses (e.g. using time points and relations among them, attaching time information to atoms instead of rules and so on).

In this perspective, we are studying the possibility of replacing time intervals with time points and constraints over them. It seems possible to adapt our approach for a constraint language with composition operators. An interval $[t_1, t_2]$ can be modelled as the constraint $t_1 \leqslant t \leqslant t_2$ and the temporal conditions in the meta-interpreter can be transformed into equivalent constraints on time points. This representation should allow us to express periodic data. For instance, suppose that a pub is open every day from 6 pm to 12 pm. This information can be modelled by constraining the fact *open_pub* by $t_1 \geqslant 18 + 24 * n \wedge t_2 \leqslant 24 + 24 * n$, where t_1 and t_2 are the starting and ending point of the validity interval, respectively. In such a way, by varying the natural number n, we obtain the opening hours of the pub. Moreover, we would like to exploit the technology of constraint logic programming in order to get better implementations than the ones provided by meta-interpreters.

A critical point is the handling of negation. From a practical viewpoint it is sufficient to add the clause

$demo(exp,\ not\ x,\ I) \leftarrow not\ demo(exp,x,I)$

to the meta-interpreter in order to embody the negation by default of logic programming into our language, as in Frühwirth (1996) and Hrycej (1993). However, from a theoretical viewpoint, the interactions between negation by default and program composition operators is still to be fully understood. Some results on the semantics interactions between operators and negation by default are presented in Brogi et al. (1997), where, however, the handling of time is not considered. One problem related to time is that the set of points in which a negation holds is generally non-convex. In fact even if A holds in a single convex interval I, the negation of A holds both in the set of points before and after I. In our approach we can derive only convex sets. However, we can represent that a fact A (or a rule) holds in a set of disjunctive and non-meeting intervals $\{I_1, \ldots, I_n\}$ by n clauses, each having as validity interval one I_i, i.e. $A \leftarrow \Box I_i$ for $i = 1 \ldots n$. Such a representation requires further investigation because it could be source of incorrect answers as shown in Böhlen and Marti (1994).

Another research direction is about designing more powerful operators. We have two categories in mind: hierarchical operators, i.e. operators that define hierarchical relations among programs, and constraint operators, that allow one to use a program as a set of constraints acting over other programs. An operator of this kind, operating over programs without time decorations, has been presented in Aquilino *et al.* (1997).

Finally, as far as applications are concerned, we are tackling the problem of integrating our logic database technology with systems for managing geographic information. In Aquilino *et al.* (1996) a solution to the problem is proposed, but, again, without any capability of handling time, while the ability of performing spatio-temporal reasoning on geographic data is nowadays considered a must for many real applications.

Acknowledgments

We thank Paolo Baldan for his useful comments and suggestions on earlier versions of this paper, Eliana Giusti for her contribution in the initial stage of this work and the anonymous referees for their useful suggestions. This work has been partially supported by MURST.

Appendix

Proof of Lemma 2 The proof is by structural induction on $\mathscr{P}$. We distinguish the following cases.

($\mathscr{P}$ is a plain program P).

$$clause(P, (A \leftarrow B_1, \ldots, B_n), I_0) \in T_\gamma^\omega \ \wedge\ \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathscr{I} \ \wedge\ \left(\bigcap_{j=0,n} I_j\right) \text{ not empty}$$

$\Longrightarrow$ $\{P$ is a plain program and definition of T_γ and clause (13)$\}$

$$\exists K : A \leftarrow B_1, \ldots, B_n \Box K \in ground(P) \ \wedge\ I_0 \subseteq K \ \wedge\ \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathscr{I} \ \wedge\ \left(\bigcap_{j=0,n} I_j\right) \text{ not empty}$$

$\Longrightarrow$ {definition of $\mathcal{F}_P$ and $H = K \cap \left(\bigcap_{j=1,n} I_j\right)$}
$\exists H : (A, H) \in \mathcal{F}_P(\mathcal{I}) \wedge \left(\bigcap_{j=0,n} I_j\right) \subseteq H$

($\mathcal{P} = \mathcal{Q} \cup \mathcal{R}$).

$clause(\mathcal{Q} \cup \mathcal{R}, (A \leftarrow B_1, \ldots, B_n), I_0) \in T_\gamma^\omega \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathcal{I} \wedge \left(\bigcap_{j=0,n} I_j\right)$ not empty
$\Longrightarrow$ {definition of T_γ and clauses (10) and (11) }
$((\exists J_1 : clause(\mathcal{Q}, (A \leftarrow B_1, \ldots, B_n), J_1) \in T_\gamma^\omega \wedge I_0 \subseteq J_1 \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathcal{I}) \vee (\exists J_2 : clause(\mathcal{R}, (A \leftarrow B_1, \ldots, B_n), J_2) \in T_\gamma^\omega \wedge I_0 \subseteq J_2 \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathcal{I})) \wedge \left(\bigcap_{j=0,n} I_j\right)$ not empty
$\Longrightarrow$ {inductive hypothesis}
$(\exists H_1 : (A, H_1) \in \mathcal{F}_\mathcal{Q}(\mathcal{I}) \wedge (J_1 \cap \left(\bigcap_{j=1,n} I_j\right)) \subseteq H_1 \wedge I_0 \subseteq J_1) \vee$
$(\exists H_2 : (A, H_2) \in \mathcal{F}_\mathcal{R}(\mathcal{I}) \wedge (J_2 \cap \left(\bigcap_{j=1,n} I_j\right)) \subseteq H_2 \wedge I_0 \subseteq J_2)$
$\Longrightarrow$ {$(I_0 \subseteq J_1) \vee (I_0 \subseteq J_2)$}
$(\exists H_1 : (A, H_1) \in \mathcal{F}_\mathcal{Q}(\mathcal{I}) \wedge \left(\bigcap_{j=0,n} I_j\right) \subseteq H_1) \vee$
$(\exists H_2 : (A, H_2) \in \mathcal{F}_\mathcal{R}(\mathcal{I}) \wedge \left(\bigcap_{j=0,n} I_j\right) \subseteq H_2)$
$\Longrightarrow$ {set-theoretic union }
$\exists H : ((A, H) \in \mathcal{F}_\mathcal{Q}(\mathcal{I}) \vee (A, H) \in \mathcal{F}_\mathcal{R}(\mathcal{I})) \wedge \left(\bigcap_{j=0,n} I_j\right) \subseteq H$
$\Longrightarrow$ {definition of $\mathcal{F}_{\mathcal{Q}\cup\mathcal{R}}$ }
$\exists H : (A, H) \in \mathcal{F}_{\mathcal{Q}\cup\mathcal{R}}(\mathcal{I}) \wedge \left(\bigcap_{j=0,n} I_j\right) \subseteq H$

($\mathcal{P} = \mathcal{Q} \cap \mathcal{R}$). As in the previous case by using clause (12) of the meta-interpreter. □

Proof of Lemma 3: The proof is by structural induction on $\mathcal{P}$. We distinguish the following cases.
($\mathcal{P}$ is a plain program P).

$(A, I) \in \mathcal{F}_P(\mathcal{I})$
$\Longleftrightarrow$ {definition of $\mathcal{F}_P$}
$\exists (B_1, I_1), \ldots, (B_n, I_n),\ I_0 : A \leftarrow B_1, \ldots, B_n \square I_0 \in ground(P) \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathcal{I} \wedge I = \bigcap_{j=0,n} I_j \wedge I \neq \varnothing$
$\Longrightarrow$ {definition of meta-level representation}
$\exists (B_1, I_1), \ldots, (B_n, I_n),\ I_0 : clause(P, (A \leftarrow B_1, \ldots, B_n), I_0) \leftarrow I_0 \sqsubseteq I_0 \in ground(\mathcal{V}) \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathcal{I} \wedge I = \bigcap_{j=0,n} I_j$
$\Longrightarrow$ {definition and monotonicity of T_γ and $I_0 \sqsubseteq I_0$ by definition}
$\exists (B_1, I_1), \ldots, (B_n, I_n),\ I_0 : clause(P, (A \leftarrow B_1, \ldots, B_n), I_0) \in T_\gamma^\omega \wedge \{(B_1, I_1), \ldots, (B_n, I_n)\} \subseteq \mathcal{I} \wedge I = \bigcap_{j=0,n} I_j$

($\mathcal{P} = \mathcal{Q} \cup \mathcal{R}$).

$(A, I) \in \mathcal{F}_{\mathcal{Q}\cup\mathcal{R}}(\mathcal{I})$
$\Longleftrightarrow$ {definition of $\mathcal{F}_{\mathcal{Q}\cup\mathcal{R}}$}
$(A, I) \in \mathcal{F}_\mathcal{Q}(\mathcal{I}) \cup \mathcal{F}_\mathcal{R}(\mathcal{I})$
$\Longleftrightarrow$ {property of set-theoretic union}
$(A, I) \in \mathcal{F}_\mathcal{Q}(\mathcal{I}) \vee (A, I) \in \mathcal{F}_\mathcal{R}(\mathcal{I})$
$\Longrightarrow$ {inductive hypothesis}

$$
\begin{array}{l}
(\exists (B_1^1, I_1^1), \ldots, (B_n^1, I_n^1),\ I_0^1 :\ clause(\mathscr{Q}, (A \leftarrow B_1^1, \ldots, B_n^1), I_0^1) \in T_\gamma^\omega \ \wedge \\
\qquad \{(B_1^1, I_1^1), \ldots, (B_n^1, I_n^1)\} \subseteq \mathscr{I} \ \wedge\ I = \bigcap_{j=0,n} I_j^1) \ \vee \\
(\exists (B_1^2, I_1^2), \ldots, (B_m^2, I_m^2),\ I_0^2 :\ clause(\mathscr{R}, (A \leftarrow B_1^2, \ldots, B_m^2), I_0^2) \in T_\gamma^\omega \ \wedge \\
\qquad \{(B_1^2, I_1^2), \ldots, (B_m^2, I_m^2)\} \subseteq \mathscr{I} \ \wedge\ I = \bigcap_{j=0,m} I_j^2) \\
\Longrightarrow \quad \{\text{clauses (10) and (11) and } I_0^1 \sqsubseteq I_0^1 \text{ and } I_0^2 \sqsubseteq I_0^2 \text{ by definition and } T_\gamma^\omega \\
\qquad \text{is a fixpoint of } T_\gamma\ \} \\
(\exists (B_1^1, I_1^1), \ldots, (B_n^1, I_n^1),\ I_0^1 :\ clause(\mathscr{Q} \cup \mathscr{R}, (A \leftarrow B_1^1, \ldots, B_n^1), I_0^1) \in T_\gamma^\omega \ \wedge \\
\qquad \{(B_1^1, I_1^1), \ldots, (B_n^1, I_n^1)\} \subseteq \mathscr{I} \ \wedge\ I = \bigcap_{j=0,n} I_j^1) \ \vee \\
(\exists (B_1^2, I_1^2), \ldots, (B_m^2, I_m^2)\ I_0^2 :\ clause(\mathscr{Q} \cup \mathscr{R}, (A \leftarrow B_1^2, \ldots, B_m^2), I_0^2) \in T_\gamma^\omega \ \wedge \\
\qquad \{(B_1^2, I_1^2), \ldots, (B_m^2, I_m^2)\} \subseteq \mathscr{I} \ \wedge\ I = \bigcap_{j=0,m} I_j^2)
\end{array}
$$

which clearly implies the thesis.

($\mathscr{P} = \mathscr{Q} \cap \mathscr{R}$). As in the previous case by using clause (12) of the meta-interpreter. □

Proof of Lemma 5 The proof is by induction on h.
(Base case). Trivial since $T_{\gamma *}^0 = \varnothing$.
(Inductive case). Assume that

$$
\begin{array}{l}
demo(\mathscr{P}, (B_1, \ldots, B_n), I) \in T_{\gamma *}^h \Longrightarrow \exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^n, \ldots, I_{i_n}^n, q : \\
\qquad \{demo(\mathscr{P}, B_j, I_r^j) \mid j = 1..n, r = 1..i_j\} \subseteq T_\gamma^q \ \wedge \\
\qquad I \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} I_j^n\right) \ \wedge \\
\qquad \bigcup_{r=1,i_j} I_r^j \in Int \text{ for each } j = 1..n
\end{array}
$$

Then:

$$
\begin{array}{l}
\quad demo(\mathscr{P}, (B_1, \ldots, B_n), I) \in T_{\gamma *}^{h+1} \\
\Longleftrightarrow \quad \{\text{definition of } T_{\gamma *}^\alpha\} \\
\quad demo(\mathscr{P}, (B_1, \ldots, B_n), I) \in T_{\gamma *}(T_{\gamma *}^h) \\
\Longleftrightarrow \quad \{\text{definition of } T_{\gamma *}\} \\
\quad \exists\ Body :\ (demo(\mathscr{P}, (B_1, \ldots, B_n), I) \leftarrow Body) \in ground(\mathscr{V}^*) \ \wedge\ Body \subseteq T_{\gamma *}^h.
\end{array}
$$

We have three cases corresponding to the clauses (8), (9) and (16):

(*a*) $n > 1$ and $Body = demo(\mathscr{P}, (B_1, \ldots, B_r), K)$,
$demo(\mathscr{P}, (B_{r+1}, \ldots, B_n), J), I \sqsubseteq K \sqcap J$,
where $1 \leqslant r < n$;

(*b*) $n = 1$ and $Body = clause(\mathscr{P}, B \leftarrow (A_1, \ldots, A_r), K)$,
$demo(\mathscr{P}, (A_1, \ldots, A_r), J), I \sqsubseteq K \sqcap J$;

(*c*) $Body = demo(\mathscr{P}, (B_1, \ldots, B_n), K), demo(\mathscr{P}, (B_1, \ldots, B_n), J), I = K \sqcup J$.

$$
\begin{array}{l}
(a)\ \exists K, J :\ (demo(\mathscr{P}, (B_1, \ldots, B_n), I) \leftarrow demo(\mathscr{P}, (B_1, \ldots, B_r), K), \\
\quad demo(\mathscr{P}, (B_{r+1}, \ldots, B_n), J), I \sqsubseteq K \sqcap J) \in ground(\mathscr{V}^*) \ \wedge \\
\quad \{demo(\mathscr{P}, (B_1, \ldots, B_r), K), demo(\mathscr{P}, (B_{r+1}, \ldots, B_n), J)\} \subseteq T_{\gamma *}^h \ \wedge I \subseteq K \cap J \\
\Longrightarrow \quad \{\text{inductive hypothesis}\} \\
\quad \exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_{i_r}^r, \ldots, I_{i_r}^r, \ldots, I_1^n, \ldots, I_{i_n}^n, q_1, q_2 :\ \{demo(\mathscr{P}, B_j, I_s^j) \mid \\
\quad j = 1..r, s = 1..i_j\} \subseteq T_\gamma^{q_1} \ \wedge K \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_r} I_j^r\right) \ \wedge \\
\quad \bigcup_{s=1,i_j} I_s^j \in Int \text{ for each } j = 1..r \ \wedge \\
\quad \{demo(\mathscr{P}, B_j, I_s^j) \mid j = r+1..n, s = 1..i_j\} \subseteq T_\gamma^{q_2}
\end{array}
$$

$\wedge\ J \subseteq \left(\bigcup_{j=1,i_{r+1}} I_j^{r+1}\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} I_j^n\right)\ \wedge$
$\bigcup_{s=1,i_j} I_s^j \in Int$ for each $j = r+1..n\ \wedge\ I \subseteq K \cap J$

$\Longrightarrow$ {$I \subseteq K \cap J$, associativity of $\cap$, $q = max\{q_1, q_2\}$ and monotonicity of T_γ }
$\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^n, \ldots, I_{i_n}^n, q :\ \{demo(\mathscr{P}, B_j, I_s^j) \mid j = 1..n, s = 1..i_j\} \subseteq T_\gamma^q\ \wedge$
$I \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} I_j^n\right)\ \wedge\ \bigcup_{s=1,i_j} I_s^j \in Int$ for each $j = 1..n$

(*b*) $\exists K, J :\ (demo(\mathscr{P}, B, I) \leftarrow (clause(\mathscr{P}, B \leftarrow (A_1, \ldots, A_r), K),$
$demo(\mathscr{P}, (A_1, \ldots, A_r), J), I \sqsubseteq K \sqcap J) \in ground(\mathscr{V}^*)\ \wedge$
$\{clause(\mathscr{P}, B \leftarrow (A_1, \ldots, A_r), K), demo(\mathscr{P}, (A_1, \ldots, A_r), J)\} \subseteq T_\gamma^h{}_*\ \wedge I \subseteq K \cap J$

$\Longrightarrow$ {inductive hypothesis}
$\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^r, \ldots, I_{i_r}^r, q :\ \{demo(\mathscr{P}, A_j, I_s^j) \mid j = 1..r, s = 1..i_j\} \subseteq T_\gamma^q\ \wedge$
$J \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_r} I_j^r\right)\ \wedge\ \bigcup_{s=1,i_j} I_s^j \in Int$ for each $j = 1..r\ \wedge$
$I \subseteq K \cap J\ \wedge\ clause(\mathscr{P}, B \leftarrow (A_1, \ldots, A_r), K) \in T_\gamma^h{}_*$

$\Longrightarrow$ {apply $r-1$ times clause (8) and let $m = q + r - 1$}
$\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^r, \ldots, I_{i_r}^r, m :$
$\{demo(\mathscr{P}, (A_1, \ldots, A_r), \prod_{j=1,r} I_{t_j}^j) \mid j = 1..r, t_j = 1..i_j,\ \bigcap_{j=1,r} I_{t_j}^j$ not empty$\} \subseteq$
$T_\gamma^m\ \wedge J \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_r} I_j^r\right)\ \wedge\ \bigcup_{s=1,i_j} I_s^j \in Int$
for each $j = 1..r\ \wedge I \subseteq K \cap J\ \wedge\ clause(\mathscr{P}, B \leftarrow (A_1, \ldots, A_r), K) \in T_\gamma^h{}_*$

$\Longrightarrow$ {distributivity of $\cap$ over $\cup$ and $\left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_r} I_j^r\right) \in Int$}
$\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^r, \ldots, I_{i_r}^r, m :$
$\{demo(\mathscr{P}, (A_1, \ldots, A_r), \prod_{j=1,r} I_{t_j}^j) \mid j = 1..r, t_j = 1..i_j, \bigcap_{j=1,r} I_{t_j}^j$ not empty$\}$
$\subseteq T_\gamma^m\ \wedge J \subseteq \bigcup\{\bigcap_{j=1,r} I_{t_j}^j \mid j = 1..r, t_j = 1..i_j, \bigcap_{j=1,r} I_{t_j}^j$ not empty$\}\ \wedge$
$\bigcup\{\bigcap_{j=1,r} I_{t_j}^j \mid j = 1..r, t_j = 1..i_j, \bigcap_{j=1,r} I_{t_j}^j$ not empty$\} \in Int\ \wedge$
$I \subseteq K \cap J\ \wedge\ clause(\mathscr{P}, B \leftarrow (A_1, \ldots, A_r), K) \in T_\gamma^h{}_*$

$\Longrightarrow$ {Lemma 4 and monotonicity of T_γ }
$\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^r, \ldots, I_{i_r}^r, m :$
$\{demo(\mathscr{P}, (A_1, \ldots, A_r), \prod_{j=1,r} I_{t_j}^j) \mid j = 1..r, t_j = 1..i_j, \bigcap_{j=1,r} I_{t_j}^j$ not empty$\}$
$\subseteq T_\gamma^m\ \wedge J \subseteq \bigcup\{\bigcap_{j=1,r} I_{t_j}^j \mid j = 1 \ldots r, t_j = 1 \ldots i_j, \bigcap_{j=1,r} I_{t_j}^j$ not empty$\}\ \wedge$
$\bigcup\{\bigcap_{j=1,r} I_{t_j}^j \mid j = 1..r, t_j = 1..i_j, \bigcap_{j=1,r} I_{t_j}^j$ not empty$\} \in Int\ \wedge$
$I \subseteq K \cap J\ \wedge\ clause(\mathscr{P}, B \leftarrow (A_1, \ldots, A_r), K) \in T_\gamma^m$

$\Longrightarrow$ {Let $H_1, \ldots, H_v$ be the elements of the set
$\{K \cap (\bigcap_{j=1,r} I_{t_j}^j) \mid j = 1 \ldots r, t_j = 1 \ldots i_j, K \cap (\bigcap_{j=1,r} I_{t_j}^j)$ not empty$\}$,
$I \subseteq K \cap J$, distributivity of $\cap$ over $\bigcup$, $K \cap J \in Int$, clause (9),
and $q' = m + 1$ }
$\exists H_1, \ldots, H_v, q' :\ \{demo(\mathscr{P}, B, H_j) \mid j = 1..v\} \subseteq T_\gamma^{q'}\ \wedge\ I \subseteq \bigcup_{j=1,v} H_j\ \wedge$
$\bigcup_{j=1,v} H_j \in Int$

(*c*) $\exists K, J :\ (demo(\mathscr{P}, (B_1, \ldots, B_n), I) \leftarrow demo(\mathscr{P}, (B_1, \ldots, B_n), K),$
$demo(\mathscr{P}, (B_1, \ldots, B_n), J), I = K \sqcup J) \in ground(\mathscr{V}^*)\ \wedge\ \{demo(\mathscr{P}, (B_1, \ldots, B_n), K),$
$demo(\mathscr{P}, (B_1, \ldots, B_n), J)\} \subseteq T_\gamma^h{}_*\ \wedge I = K \bigcup J\ \wedge\ I \in Int$

$\Longrightarrow$ {inductive hypothesis}
$\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^n, \ldots, I_{i_n}^n, \overline{I}_1^1, \ldots, \overline{I}_{i'_1}^1, \ldots, \overline{I}_1^n, \ldots, \overline{I}_{i'_n}^n, q_1, q_2 :$

$\{demo(\mathscr{P}, B_j, I_s^j) \mid j = 1..n, s = 1..i_j\} \subseteq T_\gamma^{q_1} \wedge$
$K \subseteq \left(\bigcup_{j=1,i_1} I_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i_n} I_j^n\right) \wedge \bigcup_{s=1,i_j} I_s^j \in Int$
for each $j = 1..n \wedge \{demo(\mathscr{P}, B_j, \overline{I}_s^j) \mid j = 1..n, s = 1..i'_j\} \subseteq T_\gamma^{q_2} \wedge$
$J \subseteq \left(\bigcup_{j=1,i'_1} \overline{I}_j^1\right) \cap \ldots \cap \left(\bigcup_{j=1,i'_n} \overline{I}_j^n\right) \wedge \bigcup_{s=1,i'_j} \overline{I}_s^j \in Int$ for each $j = 1..n \wedge$
$I = K \cup J \wedge I \in Int$

$\Longrightarrow$ {$I = K \cup J$ and distributivity of $\cup$ over $\cap$ and $I \in Int$,
$q = max\{q_1, q_2\}$ and monotonicity of T_γ }

$\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^n, \ldots, I_{i_n}^n, \overline{I}_1^1, \ldots, \overline{I}_{i'_1}^1, \ldots, \overline{I}_1^n, \ldots, \overline{I}_{i'_n}^n, q$:
$\{demo(\mathscr{P}, B_j, I_s^j) \mid j = 1..n, s = 1..i_j\} \subseteq T_\gamma^q \wedge$
$\{demo(\mathscr{P}, B_j, \overline{I}_s^j) \mid j = 1..n, s = 1..i'_j\} \subseteq T_\gamma^q \wedge$
$I \subseteq \left(\left(\bigcup_{j=1,i_1} I_j^1\right) \cup \left(\bigcup_{j=1,i'_1} \overline{I}_j^1\right)\right) \cap \ldots \cap \left(\left(\bigcup_{j=1,i_n} I_j^n\right) \cup \left(\bigcup_{j=1,i'_n} \overline{I}_j^n\right)\right) \wedge$
$I \in Int$

$\Longrightarrow$ {associativity of $\cup$ and $\left(\bigcup_{s=1,i_j} I_s^j\right) \cup \left(\bigcup_{s=1,i'_j} \overline{I}_s^j\right)$ is not empty,
for each $j = 1..n$, since $I \in Int$}

$\exists I_1^1, \ldots, I_{i_1}^1, \ldots, I_1^n, \ldots, I_{i_n}^n, \overline{I}_1^1, \ldots, \overline{I}_{i'_1}^1, \ldots, \overline{I}_1^n, \ldots, \overline{I}_{i'_n}^n, q$:
$\{demo(\mathscr{P}, B_j, I_s^j) \mid j = 1..n, s = 1..i_j\} \subseteq T_\gamma^q \wedge$
$\{demo(\mathscr{P}, B_j, \overline{I}_s^j) \mid j = 1..n, s = 1..i'_j\} \subseteq T_\gamma^q \wedge$
$I \subseteq (I_1^1 \cup \ldots \cup I_{i_1}^1 \cup \overline{I}_1^1 \cup \ldots \cup \overline{I}_{i'_1}^1) \cap \ldots \cap (I_1^n \cup \ldots \cup I_{i_n}^n \cup \overline{I}_1^n \cup \ldots \cup \overline{I}_{i'_n}^n)$ □

References

Abadi, M., and Manna, Z., 1989, Temporal logic programming. *Journal of Symbolic Computation*, **8:** 277–295.

Allen, J. F., 1984, Towards a general theory of action and time. *Artificial Intelligence*, **23:** 123–154.

Aquilino, D., Asirelli, P., Renso, C., and Turini, F., 1997, Applying restriction constraint to deductive databases. *Annals of Mathematics and Artificial Intelligence*, **19:** 3–25.

Aquilino, D., Renso, C., and Turini, F., 1996, Towards declarative GIS analysis. *Proocedings of the Fourth ACM Workshop on Advances in Geographic Information Systems*, edited by S. Shekhar and P. Bergougnoux, pp. 99–105.

Baudinet, M., 1989, Temporal logical programming is complete and expressive. *ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pp. 267–280.

Baudinet, M., Chomicki, J., and Wolper, P., 1993, Temporal deductive databases. *Temporal Databases: Theory, Design, and Implementation*, edited by A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev and R. Snodgrass (Benjamin/ Cummings), pp. 294–320.

Böhlen, M., and Marti, R., 1994, On the Completeness of Temporal Database Query Languages. *Temporal Logic, Proceedings of the First International Conference, ICTL'94*, No. 827, *Lecture Notes in Artificial Intelligence*, pp. 283–300,.

Bowen, K. A., and Kowalski, R. A., 1982, Amalgamating language and metalanguage in logic programming. *Logic Programming*, Vol. 16, *APIC Studies in Data Processing*, edited by K. L. Clark and S.-A. Tarnlund (New York: Academic Press), pp. 153–172.

Brogi, A., 1993, Program construction in computational logic. PhD thesis, Dipartimento di Informatica, Università di Pisa, Italy.

Brogi, A., Contiero, S., and Turini, F., 1997, Composing general logic programs. *Proceedings of the Fourth International Conference, LPNMR'97*, Vol. 1265, *Lecture Notes in Artificial Intelligence*, pp. 273–288.

Brogi, A., Mancarella, P., Pedreschi, D., and Turini, F., 1994, Modular logic programming. *ACM Transactions on Programming Languages and Systems*, **16**(4): 1361–1398.

Brogi, A., and Turini, F., 1994, Semantics of Meta-logic in an Algebra of Programs. *Proceedings of the Ninth IEEE Symposium on Logic in Computer Science* (Piscataway, NJ: IEEE Press), pp. 262–270.

Brzoska, C., 1995, Temporal logic programming with metric and past operators. *Executable Modal and Temporal Logics*, Vol. 897, *Lecture Notes in Artificial Intelligence*, edited by M. Fisher and R. Owens, pp. 21–39.

Chomicki, J., 1994, Temporal query languages: a survey. *Lecture Notes in Artificial Intelligence*, Vol. 827, pp. 506–534.

Chomicki, J., and Imielinski, T., 1988, Temporal deductive databases and infinite objects. *Proceedings of ACM SIGACT/ SIGMOD Symposium on Principles of Database Systems*, pp. 61–73.

Fisher, M., and Owens, R. (editors), 1995, *Executable Modal and Temporal Logics*, Vol. 897, *Lecture Notes in Artificial Intelligence* (Berlin: Springer Verlag).

Frühwirth, T., 1996, Temporal annotated constraint logic programming. *Journal of Symbolic Computation*, **22:** 555–583.

Gabbay, D. M., 1987, Modal and temporal logic programming. *Temporal Logics and Their Applications*, edited by A. Galton (New York: Academic Press), pp. 197–237.

Gabbay, D. M., and McBrien, P., 1991, Temporal logic & historical databases. *Proceedings of the Seventeenth International Conference on Very Large Databases*, September, pp. 423–430, .

Hill, P. M., and Lloyd, J. W., 1989, Analysis of metaprograms. *Metaprogramming in Logic Programming*, edited by H. D. Abramson and M. H. Rogers (Cambridge, MA: The MIT Press), pp. 23–52.

Hrycej, T., 1993, A temporal extension of Prolog. *Journal of Logic Programming*, **15**(1& 2): 113–145.

Jaffar, J., and Maher, M. J., 1994, Constraint logic programming: a survey. *The Journal of Logic Programming*, **19 & 20:** 503–582.

Kanellakis, P. C., Kuper, G. M., and Revesz, P. Z., 1995, Constraint query languages. *Journal of Computer and System Sciences*, **51**(1): 26–52.

Koubarakis, M., 1994, Database models for infinite and indefinite temporal information. *Information Systems*, **19**(2): 141–173.

Kowalski, R. A., 1992, Database updates in the Event Calculus. *Journal of Logic Programming*, **12:** 121–146.

Kowalski, R. A., and Kim, J. S., 1991, A metalogic programming approach to multi-agent knowledge and belief. *Artificial Intelligence and Mathematical Theory of Computation*, edited by V. Lifschitz (New York: Academic Press).

Kowalski, R. A., and Sergot, M. J., 1986, A logic-based calculus of events. *New Generation Computing*, **4**(1): 67–95.

Mancarella, P., Raffaetà, A., and Turini, F., 1997, Time in a multi-theory logical framework. *Proceedings of the Fourth International Workshop on Temporal Representation and Reasoning (TIME'97)*, edited by R. Morris and L. Khatib (IEEE Computer Society Press), pp. 62–68.

Orgun, M. A., 1996, On temporal deductive databases. *Computational Intelligence*, **12**(2): 235–259.

Orgun, M. A., and Ma, W., 1994, An overview of temporal and modal logic programming. *Lecture Notes in Artificial Intelligence*, Vol. 827, pp. 445–479.

Orgun, M. A., and Wadge, W. W., 1988, A theoretical basis for intensional logic programming. *Proceedings of the 1988 International Symposium on Lucid and Intensional Programming*, pp. 33–49.

Rolston, D. W., 1986, *Chronolog: A Pure Tense-Based Infinite-Object Programming Language* (Arizona, USA: Department of Computer Science and Engineering, Arizona State University).

Sakuragawa, T., 1987, Temporal prolog. *Proceedings of RIMS Conference on Software Science and Engineering* (Berlin: Springer-Verlag).

Snodgrass, R., 1992, Temporal databases. *Proceedings of the International Conference on GIS - From Space to Territory: Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pp. 22–64.

Sripada, S. M., 1988, A logical framework for temporal deductive databases. *Proceedings of the Very Large Databases Conference*, pp. 171–182.

Sripada, S. M., 1991, Temporal reasoning in deductive databases. PhD thesis, Department of Computing Imperial College of Science & Technology, UK.

Sterling, L., and Shapiro, E., 1986, *The Art of Prolog* (Cambridge, MA: The MIT Press).

Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R. (editors), 1993, *Temporal Databases: Theory, Design, and Implementation* (Benjamin/ Cummings).

Zaniolo, C., Arni, N., and Ong, K., 1993, Negation and aggregates in recursive rules: the LDL++ Approach. *International conference on Deductive and Object-Oriented Databases (DOOD'93)*, Vol. 760, *Lecture Notes in Computer Science*.